

Asmedia ASM1064/116x Series PCI-E to SATA AHCI Controller

System BIOS Programming Note

Revision 1.5

2019/11/25

ASMEDIA TECHNOLOGY INC.

Revision History

Rev	Date	Author	Description
1.5	2019/11/25	Jesse Chang	Add section 9. PCI Memory Space Access Constraint
1.4	2019/11/12	Jesse Chang	1. Add section 7. PCI Express Capability Registers 2. Add section 8. Use I ² C to Access Internal Registers
1.3	2019/11/01	Jesse Chang	1. Modify title to System BIOS Programming Note 2. Add section 5. GPIO Registers 3. Add section 6. LED Registers
1.2	2019/10/04	Jesse Chang	Modify ASM1062 to ASM1062A
1.1	2019/09/27	Jesse Chang	1. Correct SVID/SSID default values 2. Add section 3.4. SATA Transmitter Registers
1.0	2019/09/25	Jesse Chang	Initial version

Contents

Revision History	2
Contents	3
1. Introduction	5
2. Registers Space	6
2.1. PCIe Configuration Space	6
2.2. Private Control Space (BAR0 MMIO)	6
2.3. AHCI Control Space (BAR5 MMIO)	9
2.3.1. Memory Read/Write	9
2.3.2. Index-Data Pair	9
3. Registers Setting	14
3.1. Program PCI SVID and SSID	14
3.2. Disable Safety Removal for AHCI Mode	14
3.3. SATA SSC Control	16
3.4. SATA Transmitter Registers	17
4. I ² C Controller	20
4.1. Write Byte Example	27
4.2. Read Byte Example	28
5. GPIO Registers	30
6. LED Registers	34
7. PCI Express Capability Registers	47
7.1. PCI Express Capability List Register	47
7.2. PCI Express Capabilities Register	47
7.3. Device Capabilities Register	48
7.4. Device Control Register	48
7.5. Device Status Register	49
7.6. Link Capabilities Register	49
7.7. Link Control Register	49
7.8. Link Status Register	50
7.9. Link Capabilities 2 Register	51
7.10. Advanced Error Reporting (AER) Capability Register	51
7.10.1. Advanced Error Reporting Extended Capability Header	51
7.10.2. Uncorrectable Error Status Register	52
7.10.3. Uncorrectable Error Mask Register	52

7.10.4. Uncorrectable Error Severity Register	53
7.10.5. Correctable Error Status Register	53
7.10.6. Correctable Error Mask Register	54
7.10.7. Advanced Error Capabilities and Control Register	54
7.10.8. Header Log Register	55
8. Use I ² C to Access Internal Registers	56
8.1. Internal Registers Map	56
8.2. Read/Write Protocols	56
9. PCI Memory Space Access Constraint	58

ASMedia Confidential

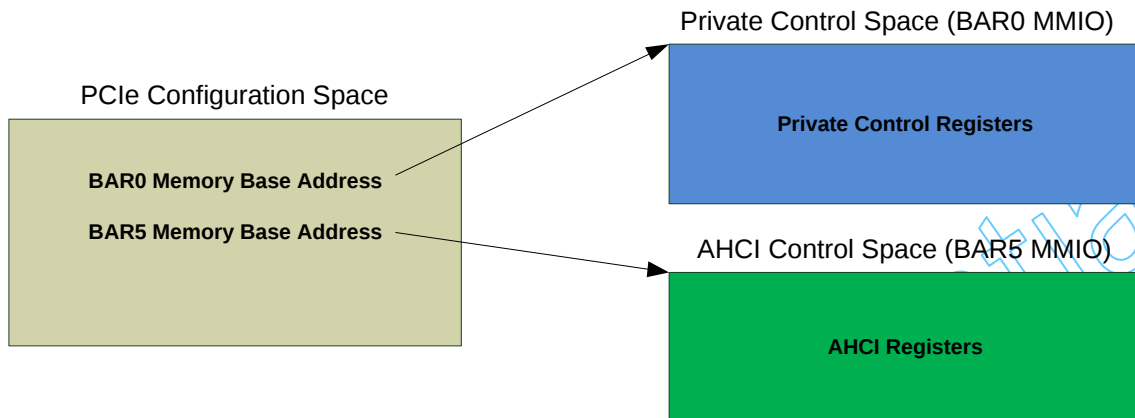
1. Introduction

Asmedia ASM1064/116x series are PCIE to SATA AHCI controllers and supports native Port Multiplier function. This document describes how to program and which registers are needed to program by System BIOS. Next table lists the brief features of ASM1064/116x series controllers. Please refer its datasheet for the detail.

Table1 ASM1064/116x features

Product	ASM1062A	ASM1064	ASM1164	ASM1165	ASM1166
VID	1B21h	1B21h	1B21h	1B21h	1B21h
DID	1062h	1064h	1164h	1165h	1166h
PCie	Gen3 x1	Gen3 x1	Gen3 x2	Gen3 x2	Gen3 x2
SATA / Port	Gen3 / 2 ports	Gen3 / 4 ports	Gen3 / 4 ports	Gen3 / 5 ports	Gen3 / 6 ports
LED	2	4	4	5	6
DevsIp pin	2	4	4	5	6
GPIO	2	6	0	4	6

2. Registers Space



There are three Register Spaces – PCIe Configuration Space, Private Control Space, and AHCI Control Space. Following sections describe each Register Space, separately.

2.1. PCIe Configuration Space

Standard PCI Express Configuration Space follows “PCI Express Base Specification Revision 3.1”. The access method is not described in this document. Please refer to PCI Express Base Specification for the detail.

2.2. Private Control Space (BAR0 MMIO)

The registers in Private Control space can be accessed by normal memory read/write only after BAR0 (PCI Rx10 ~ Rx14) MMIO address is assigned and Memory Space enabled (PCI Rx04[1]). Don’t program registers which are not described in this document. Which registers can be programmed is described in “Registers Setting” section. Undetermined behavior may occur if registers are not programmed correctly.

Followings are the same code to access Private Control registers:

```

#define PCI_BAR0                0x10                //PCI offset 0x10 ~ 0x13: BAR0
#define PCI_ADDRESS_MEMORY_TYPE_MASK 0x00000006L    //bit 3, 2
#define PCI_ADDRESS_MEMORY_PREFETCHABLE 0x00000008L //bit 4
  
```

//*****

```

//Get Private Control Base Address (BAR0)
// Input:   PciBus      - PCI bus number of ASM116x
//          PciDev      - PCI device number of ASM116x
//          PciFun      - PCI function number of ASM116x
// Output:   Private Control Base Address (BAR0)
//*****
DWORD ASM116xGetPrivateControlBaseAddress(BYTE PciBus, BYTE PciDev, BYTE PciFun)
{
    DWORD    Bar0;

    //Get Private Control base address BAR0
    Bar0 = PciReadDword(PciBus, PciDev, PciFun, PCI_BAR0);
    Bar0 = Bar0 & (~PCI_ADDRESS_MEMORY_TYPE_MASK) & (~PCI_ADDRESS_MEMORY_PREFETCHABLE);
    return Bar0;
}

//*****
// Private Control Read BYTE
// Input:   BaseAddr    - Private Control Base Address (BAR0)
//          Register     - Register offset
// Output:   Read value
//*****
BYTE ASM116xPrivateControlReadByte(DWORD BaseAddr, WORD Register)
{
    DWORD    MemAddr;        //Physical memory address
    PBYTE    pByte;
    BYTE     bValue;

    MemAddr = BaseAddr + Register;
    pByte = (PBYTE)MemAddr;
    bValue = *pByte;
    return bValue;
}

//*****
// Private Control Read WORD
// Input:   BaseAddr    - Private Control Base Address (BAR0)
//          Register     - Register offset
// Output:   Read value
//*****
WORD ASM116xPrivateControlReadWord(DWORD BaseAddr, WORD Register)
{
    DWORD    MemAddr;        //Physical memory address
    PWORD    pWord;
    WORD     wValue;

    MemAddr = BaseAddr + Register;
    pWord = (PWORD)MemAddr;
    wValue = *pWord;
    return wValue;
}

```

```

//*****
// Private Control Read DWORD
// Input:      BaseAddr  - Private Control Base Address (BAR0)
//            Register    - Register offset
// Output:      Read value
//*****
DWORD ASM116xPrivateControlReadDword(DWORD BaseAddr, WORD Register)
{
    DWORD    MemAddr;        //Physical memory address
    PDWORD    pDword;
    DWORD    dValue;

    MemAddr = BaseAddr + Register;
    pDword = (PDWORD)MemAddr;
    dValue = *pDword;
    return dValue;
}

//*****
// Private Control Write BYTE
// Input:      BaseAddr  - Private Control Base Address (BAR0)
//            Register    - Register offset
//            bValue      - Written value
// Output:      none
//*****
VOID ASM116xPrivateControlWriteByte(DWORD BaseAddr, WORD Register, BYTE bValue)
{
    DWORD    MemAddr;        //Physical memory address
    PBYTE    pByte;

    MemAddr = BaseAddr + Register;
    pByte = (PBYTE)MemAddr;
    *pByte = bValue;
    return;
}

//*****
// Private Control Write WORD
// Input:      BaseAddr  - Private Control Base Address (BAR0)
//            Register    - Register offset
//            wValue      - Written value
// Output:      none
//*****
VOID ASM116xPrivateControlWriteWord(DWORD BaseAddr, WORD Register, WORD wValue)
{
    DWORD    MemAddr;        //Physical memory address
    PWORD    pWord;

    MemAddr = BaseAddr + Register;
    pWord = (PWORD)MemAddr;
    *pWord = wValue;
    return;
}

```



```

//*****
// Private Control Write DWORD
// Input:   BaseAddr   - Private Control Base Address (BAR0)
//          Register    - Register offset
//          dValue      - Written value
// Output:  none
//*****
VOID ASM116xPrivateControlWriteDword(DWORD BaseAddr, WORD Register, DWORD dValue)
{
    DWORD    MemAddr;          //Physical memory address
    PDWORD    pDword;

    MemAddr = BaseAddr + Register;
    pDword = (PDWORD)MemAddr;
    *pDword = dValue;
    return;
}

```

2.3. AHCI Control Space (BAR5 MMIO)

There are two access methods that can be used to access this space – Memory Read/Write and Index-Data Pair. The registers in this space are standard AHCI registers. Please refer to “Serial ATA Advanced Host Controller Interface (AHCI) 1.3.1” for the detail.

2.3.1. Memory Read/Write

The registers in AHCI Control space can be accessed by normal memory read/write after BAR5 (PCI Rx24 ~ Rx27) MMIO address is assigned and Memory Space enabled (PCI Rx04[1]).

2.3.2. Index-Data Pair

The registers in AHCI Control space by private Index-Data Pair method. It accesses AHCI Control space through PCI Configuration Space offset Rx00 ~ Rx07 and does not depend on PCI BAR 5 memory base address and Memory Space enabled bit.

PCI Configuration Space offset – D0h ~ D3h

Register Name: INDEX

Size: 32 bits

Bit	Attrib	Description	Default	Mnemonic
31:16	Rsvd	Reserved	0	
15:2	RW	Indirect access address	0	
1:0	Rsvd	Reserved	0	

PCI Configuration Space offset – D4h ~ D7h

Register Name: DATA

Size: 32 bits

Bit	Attrib	Description	Default	Mnemonic
31:0	RW	Indirect access data This Data register is a “window” through which data is read or written to the AHCI memory mapped registers. A read or write to this Data register triggers a corresponding read or write to the memory mapped register pointed to by the Index register. The Index register must be setup prior to the read or write to this Data register. Note that a physical register is not actually implemented as the data is actually stored in the memory mapped registers. Since this is not a physical register, the “default” value is the same as the default value of the register pointed to by Index.	0	

Following are the same code for Index-Data Pair method to access AHCI registers:

```
#define ASM116x_MMIO_INDEX      0xD0          //PCI offset 0xD0 ~ 0xD3: Index
#define ASM116x_MMIO_DATA      0xD4          //PCI offset 0xD4 ~ 0xD7: Data

//*****
// Index-Data Pair Read BYTE
// Input:   PciBus    - PCI bus number of ASM116x
//          PciDev    - PCI device number of ASM116x
//          PciFun    - PCI function number of ASM116x
//          Register  - Register offset
// Output:  Read value
//*****
BYTE ASM116xIdpReadByte(BYTE PciBus, BYTE PciDev, BYTE PciFun, DWORD Register)
{
    DWORD    Index;
    BYTE     bValue;
    BYTE     Offset;

    //Index shall be DWORD alignment
    Index = Register & (~0x03L);
    Offset = Register % 4;
    //Write Index
    PciWriteDword(PciBus, PciDev, PciFun, ASM116x_MMIO_INDEX, Index);
    //Read BYTE Data
    bValue = PciReadByte(PciBus, PciDev, PciFun, ASM116x_MMIO_DATA + Offset);
    return bValue;
}

//*****
// Index-Data Pair Read WORD
// Input:   PciBus    - PCI bus number of ASM116x
//          PciDev    - PCI device number of ASM116x
//          PciFun    - PCI function number of ASM116x
//          Register  - Register offset
// Output:  Read value
// Note:   Register offset shall be WORD alignment
//*****
```

```

WORD ASM116xIdpReadWord(BYTE PciBus, BYTE PciDev, BYTE PciFun, DWORD Register)
{
    DWORD    Index;
    WORD     wValue;
    BYTE     Offset;

    //Index shall be DWORD alignment
    Index = Register & (~0x03L);
    Offset = Register % 4;
    //Write Index
    PciWriteDword(PciBus, PciDev, PciFun, ASM116x_MMIO_INDEX, Index);
    //Read WORD Data
    wValue = PciReadWord(PciBus, PciDev, PciFun, ASM116x_MMIO_DATA + Offset);
    return wValue;
}

//*****
// Index-Data Pair Read DWORD
// Input:    PciBus    - PCI bus number of ASM116x
//           PciDev    - PCI device number of ASM116x
//           PciFun    - PCI function number of ASM116x
//           Register  - Register offset
// Output:   Read value
// Note:     Register offset shall be DWORD alignment
//*****
DWORD ASM116xIdpReadDword(BYTE PciBus, BYTE PciDev, BYTE PciFun, DWORD Register)
{
    DWORD    Index;
    DWORD    dValue;

    //Index shall be DWORD alignment
    Index = Register & (~0x03L);
    //Write Index
    PciWriteDword(PciBus, PciDev, PciFun, ASM116x_MMIO_INDEX, Index);
    //Read DWORD Data
    dValue = PciReadDword(PciBus, PciDev, PciFun, ASM116x_MMIO_DATA);
    return dValue;
}

//*****
// Index-Data Pair Write BYTE
// Input:    PciBus    - PCI bus number of ASM116x
//           PciDev    - PCI device number of ASM116x
//           PciFun    - PCI function number of ASM116x
//           Register  - Register offset
//           bValue    - Written value
// Output:   none
//*****
VOID ASM116xIdpWriteByte(BYTE PciBus, BYTE PciDev, BYTE PciFun, DWORD Register, BYTE bValue)
{
    DWORD    Index;
    BYTE     Offset;
  
```

```

    //Index shall be DWORD alignment
    Index = Register & (~0x03L);
    Offset = Register % 4;
    //Write Index
    PciWriteDword(PciBus, PciDev, PciFun, ASM116x_MMIO_INDEX, Index);
    //Write BYTE Data
    PciWriteByte(PciBus, PciDev, PciFun, ASM116x_MMIO_DATA + Offset, bValue);
    return;
}

//*****
// Index-Data Pair Write WORD
// Input:   PciBus   - PCI bus number of ASM116x
//          PciDev   - PCI device number of ASM116x
//          PciFun   - PCI function number of ASM116x
//          Register - Register offset
//          wValue   - Written value
// Output:  none
// Note:    Register offset shall be WORD alignment
//*****
VOID ASM116xIdpWriteWord(BYTE PciBus, BYTE PciDev, BYTE PciFun, DWORD Register, WORD wValue)
{
    DWORD    Index;
    BYTE     Offset;

    //Index shall be DWORD alignment
    Index = Register & (~0x03L);
    Offset = Register % 4;
    //Write Index
    PciWriteDword(PciBus, PciDev, PciFun, ASM116x_MMIO_INDEX, Index);
    //Write WORD Data
    PciWriteWord(PciBus, PciDev, PciFun, ASM116x_MMIO_DATA + Offset, wValue);
    return;
}

//*****
// Index-Data Pair Write DWORD
// Input:   PciBus   - PCI bus number of ASM116x
//          PciDev   - PCI device number of ASM116x
//          PciFun   - PCI function number of ASM116x
//          Register - Register offset
//          dValue   - Written value
// Output:  none
// Note:    Register offset shall be DWORD alignment
//*****
VOID ASM116xIdpWriteDword(BYTE PciBus, BYTE PciDev, BYTE PciFun, DWORD Register, WORD dValue)
{
    DWORD    Index;

    //Index shall be DWORD alignment
    Index = Register & (~0x03L);
    //Write Index
    PciWriteDword(PciBus, PciDev, PciFun, ASM116x_MMIO_INDEX, Index);

```

```
//Write DWORD Data  
PciWriteDword(PciBus, PciDev, PciFun, ASM116x_MMIO_DATA, dValue);  
return;  
}
```

ASMedia Confidential

3. Registers Setting

All registers described in this section that System BIOS want to set, must be set when cold boot, warm boot, S3 resume, S4 resume and before assigning PCI resources.

3.1. Program PCI SVID and SSID

The default values of ASM116x SVID (PCI Rx2C ~ 2D) and SSID (PCI Rx2E ~ 2F) are 116xh and can be modified by Private Control Rx194A ~ 194B and Rx1948 ~ 1949.

BAR0 MMIO offset – 1948h

Register Name: PCI_SSID

Size: 8 bits

Bit	Attrib	Description	Default	Mnemonic
15:0	RW	Subsystem ID	2116h	

BAR0 MMIO offset – 194Ah

Register Name: PCI_SVID

Size: 8 bits

Bit	Attrib	Description	Default	Mnemonic
15:0	RW	Subsystem Vendor ID	2116h	

Sample code:

```

//*****
// Set SVID, SSID
// Input:   BaseAddr - BAR0 Base Address
//          Svid     - SVID to be set
//          Ssid     - SSID to be set
// Output:  none
// Note:
//*****
VOID ASM116xSetSvidSsid(DWORD BaseAddr, WORD Svid, WORD Ssid)
{
    //Set SVID
    ASM116xPrivateControlWriteWord(BaseAddr, 0x194A, Svid);
    //Set SSID
    ASM116xPrivateControlWriteWord(BaseAddr, 0x1948, Ssid);
    return;
}

```

3.2. Disable Safety Removal for AHCI Mode

The Hard Disk Drive (HDD) which connected on SATA port will be displayed on Windows Notification Area and can be safely removed when use Windows inbox driver. Windows inbox AHCI

driver depends on AHCI HBA Capabilities register and each AHCI Port Command and Status register to determine whether displays the HDD on Notification Area.

When HBA Capabilities register bit 5 (CAP.SXS) is 1 that indicates the HBA supports External SATA. When Port Command and Status register bit 18 (PxCMD.HPCP) is 1 that indicates the SATA port supports Hot-Plug. If CAP.SXS or PxCMD.HCPC is 1, Windows will display the HDD which connect on this AHCI SATA port. To eliminate HDD displayed on Notification Area, the HBA CAP.SXS and PxCMD.HCPC bit of every port must be set to 0. Please refer to AHCI specification "section 3.1.1 Offset 00h: CAP – HBA Capabilities" and "section 3.3.7 Offset 18h: PxCMD – Port x Command and Status"

AHCI specification defines that CAP.SXS and PxCMD.HPCP are read only bits. They can't be modified by software. Other assistant registers are provided to modify these AHCI read only bits – BAR0 MMIO Rx1F00[5] for modifying CAP.SXS, BAR0 MMIO Rx1FC8[3] ~ Rx1FCD[3] for modifying Port 0 ~ 5 PxCMD.HCPC and BAR0 MMIO Rx1FC8[6] ~ Rx1FCD[6] for modifying Port 0 ~ 5 PxCMD.ESP.

To disable eSATA and hot-plug for AHCI Mode, please set CAP.SXS = 0, PxCMD.HPCP = 0, and PxCMD.ESP = 0.

BAR0 MMIO offset – 1F00h

Register Name: HBA_CAP

Size: 32 bits

Bit	Attrib	Description	Default	Mnemonic
31:0	RW	AHCI HBA capability backdoor	0xef36_ff3f	

BAR0 MMIO offset – 1FC8h, 1FC9h, (ASM1062A/1064/1164/1165/1166 Port 0, 1)

1FCAh, 1FCBh, (ASM1064/1164/1165/1166 port 2, 3)

1FCCh, (ASM1165/1166 port 4)

1FCDh (ASM1166 port 5)

Register Name: BKDOOR_P0/1/2/3/4/5

Size: 8 bits

Bit	Attrib	Description	Default	Mnemonic
7	RW	FIS-based switching capable port (FBSCP) backdoor	0	
6	RW	External SATA port 0(ESP) backdoor	1	
5	RW	Cold presence detection (CPD) backdoor	0	
4	RW	Mechanical presence switch attached to port (MPSP) backdoor	0	
3	RW	Hot plug capable port (HPCP) backdoor	1	
2	RW	Reserved	0	
1	RW	Mechanical presence switch state (MPSS) backdoor	0	
0	RW	Power on device (POD) backdoor	1	

Sample code:

```
//*****
// Disable safety removal
// Input:   BaseAddr   - BAR0 Base Address
// Output:   none
// Note:
//*****
VOID ASM116xDisableSafetyRemoval(DWORD BaseAddr)
{
    BYTE    bData;
    BYTE    PortNumber;

    bData = ASM116xPrivateControlReadByte(BaseAddr, 0x1F00);
    bData &= ~0x20;    //bit 5 = 0
    ASM116xPrivateControlWriteByte(BaseAddr, 0x1F00, bData);
    // BAR0 MMIO Rx1FC8[3] ~ Rx1FCD[3] = 0 - Set Port 0 ~ 5 PxCMD.HPCP = 0
    // BAR0 MMIO Rx1FC8[6] ~ Rx1FCD[6] = 0 - Set Port 0 ~ 5 PxCMD.ESP = 0
    for(PortNumber = 0; PortNumber < 6; PortNumber++)
    {
        bData = ASM116xPrivateControlReadByte(BaseAddr, 0x1FC8 + PortNumber);
        bData &= ~0x48;    //bit 3 = 0, bit 6 = 0
        ASM116xPrivateControlWriteByte(BaseAddr, 0x1FC8 + PortNumber, bData);
    }
    return;
}
```

3.3. SATA SSC Control

SATA SSC Control can be set by following steps.

1. Ensure SSC is disabled
Set BAR0 MMIO Rx198C[0] = 0
2. Set SSC Mode
Set BAR0 MMIO Rx198C[3:1] to value wanted to set
3. Enable SSC
Set BAR0 MMIO Rx198C[0] = 1

BAR0 MMIO offset – 198Ch

Register Name: SSCG_CTL

Size: 8 bits

Bit	Attrib	Description	Default	Mnemonic
7	Rsvd	Reserved	0	
6	RW	Power down SSCPLL	0	
5:4	RW	SSCPLL bandwidth 0: 10uA 1: 20uA 2~3: reserved	0	

3:1	RW	SSC mode 000b: 0~~4000ppm, 1:1 modulation 001b: 0~~2000ppm, 1:1 modulation 010b: 0~~5000ppm, 1:1 modulation 011b: 0~~1250ppm, 1:1 modulation 100b: -500~~4500ppm, 1:1 modulation 101b: -250~~4250ppm 110b: -250~~2250ppm 111b: reserved SSC mode can not be changed when SSC is enabled.	3'101b	
0	RW	SSC enable	0	

Sample code:

```

//*****
// SATA SSC Control
// Input:   BaseAddr   - BAR0 Base Address
// Output:   none
// Note:
//*****
VOID ASM116xSataSscControl(DWORD BaseAddr)
{
    BYTE    bData;

    //Ensure SSC is disabled: Set BAR0 MMIO Rx198C[0] = 0
    bData = ASM116xPrivateControlReadByte(BaseAddr, 0x198C);
    bData &= ~0x01;    //bit 0 = 0
    ASM116xPrivateControlWriteByte(BaseAddr, 0x198C, bData);
    //Set SSC Mode: Set BAR0 MMIO Rx198C[3:1] to value wanted to set
    //    Set 3'101b: -250~~-4250ppm
    //Enable SSC: Set BAR0 MMIO Rx198C[0] = 1
    bData = ASM116xPrivateControlReadByte(BaseAddr, 0x198C);
    bData &= ~0x0F;    //bit [3:0] = 0h
    bData |= 0x0B;    //bit[3:1] = 3'101b, bit 0 = 1
    ASM116xPrivateControlWriteByte(BaseAddr, 0x198C, bData);
    return;
}

```

3.4. SATA Transmitter Registers

The SATA transmitter (Tx) signal can be adjusted by following registers.

Important Note: Don't access Tx registers which SATA port is not exist. Otherwise, the behavior is not define and system will hang. Example: Don't access BAR0 MMIO offset 0922h and 0B22h on ASM1062A/1064/1164.

BAR0 MMIO offset – **0122h, 0322h, (ASM1062A/1064/1164/1165/1166 Port 0, 1)**
 0522h, 0722h, (ASM1064/1164/1165/1166 port 2, 3)
 0922h, (ASM1165/1166 port 4)
 0B22h (ASM1166 port 5)

Register Name: SATA_PHY22_P0/1/2/3/4/5

Size: 8 bits

Bit	Attrib	Description	Default	Recommend
7:4	RW	SATA Gen 1 Transmitter driver pre/de-emphasis or slew rate level control	0h	0h
3:0	RW	SATA Gen 1 Transmitter driver current control	0h	0h

BAR0 MMIO offset – 0123h, 0323h, (ASM1062A/1064/1164/1165/1166 Port 0, 1)
0523h, 0723h, (ASM1064/1164/1165/1166 port 2, 3)
0923h, (ASM1165/1166 port 4)
0B23h (ASM1166 port 5)

Register Name: SATA_PHY23_P0/1/2/3/4/5

Size: 8 bits

Bit	Attrib	Description	Default	Recommend
7:4	RW	SATA Gen 2 Transmitter driver pre/de-emphasis or slew rate level control	2h	2h
3:0	RW	SATA Gen 2 Transmitter driver current control	4h	4h

BAR0 MMIO offset – 0124h, 0324h, (ASM1062A/1064/1164/1165/1166 Port 0, 1)
0524h, 0724h, (ASM1064/1164/1165/1166 port 2, 3)
0924h, (ASM1165/1166 port 4)
0B24h (ASM1166 port 5)

Register Name: SATA_PHY24_P0/1/2/3/4/5

Size: 8 bits

Bit	Attrib	Description	Default	Recommend
7:4	RW	SATA Gen 3 Transmitter driver pre/de-emphasis or slew rate level control	3h	5h
3:0	RW	SATA Gen 3 Transmitter driver current control	8h	8h

BAR0 MMIO offset – 0125h, 0325h, (ASM1062A/1064/1164/1165/1166 Port 0, 1)
0525h, 0725h, (ASM1064/1164/1165/1166 port 2, 3)
0925h, (ASM1165/1166 port 4)
0B25h (ASM1166 port 5)

Register Name: SATA_PHY25_P0/1/2/3/4/5

Size: 8 bits

Bit	Attrib	Description	Default	Recommend
7	RW	Register Control For Function Test 0: to have normal swing; 1: to have 1/2 swing	0	0
6	RW	Transmitter Driver Signaling Control Select 1: fine tune skew of TX signaling	0	0
5:3	RW	Reserved	0	0
2	RW	SATA Gen 3 Pre/de-emphasis Function	1	1

1	RW	SATA Gen 2 Pre/de-emphasis Function	1	1
0	RW	SATA Gen 1 Pre/de-emphasis Function	0	0

ASMedia Confidential

4. I²C Controller

The ASM116x contain an I²C-Bus Host interface that allows processor to communicate with I²C slaves which compatible with "THE I²C-BUS SPECIFICATION VERSION 2.1". Followings are the registers description of I²C-Bus Host interface.

BAR0 MMIO offset – 1A00h

Register Name: I2C Control 0

Size: 8 bits

Bit	Attrib	Description	Default	Mnemonic
7:2	Rsvd	Reserved	0	
1	RWC	Timeout Write 1 to tell HW that I2C interface timeout. Then HW changes the state to receive the "Stop" command and clears the "Run" bit. This bit is auto-cleared after operation done.	0	is_timeout
0	RWC	Run Write 1 to trigger I2C operation in master mode. This bit is auto-cleared after operation done and all interrupt statuses are cleared. Please make sure this bit is '0' before you send the next command.	0	ms_run

BAR0 MMIO offset – 1A01h

Register Name: I2C Control 1

Size: 8 bits

Bit	Attrib	Description	Default	Mnemonic
7:2	Rsvd	Reserved	0	
3:2	RW	Hold time control 00: Standard Mode 01: Fast Mode 10: Fast+ Mode 11: Set Zero	2'10b	hd_dat_mode
1:0	RW	Drive mode 00: Standard Mode, clock below 100KHz, include 100KHz 01: Fast Mode, clock between 100 ~ 400KHz, include 400KHz 10: Fast+ Mode, clock above 400KHz 11: Reserved.	0	bus_drv_mode

Note: Don't change default value of this register if slave device can support 100KHz.

BAR0 MMIO offset – 1A02h

Register Name: I2C Clock High

Size: 8 bits

Bit	Attrib	Description	Default	Mnemonic
7:0	RW	Clock High Period	57h	clk_hi_prd

Note: Don't change default value of this register if slave device can support 100KHz.

BAR0 MMIO offset – 1A03h

Register Name: I2C Clock Low

Size: 8 bits

Bit	Attrib	Description	Default	Mnemonic
7:0	RW	Clock Low Period	33h	clk_low_prd

Note: Don't change default value of this register if slave device can support 100KHz.

BAR0 MMIO offset – 1A04h

Register Name: I2C Data

Size: 8 bits

Bit	Attrib	Description	Default	Mnemonic
7:0	RW	Data Data/Address byte transmitted by I2C master mode. In the case of address byte, bit[0] is R/W bit.	0	rw_dat

BAR0 MMIO offset – 1A05h

Register Name: I2C Order

Size: 8 bits

Bit	Attrib	Description	Default	Mnemonic
7:2	Rsvd	Reserved	0	
1:0	RW	Data order 00: Start + Data 01: Data 10: Data + Stop 11: Stop	0	dat_order

BAR0 MMIO offset – 1A06h

Register Name: I2C Interrupt Enable

Size: 8 bits

Bit	Attrib	Description	Default	Mnemonic
7	Rsvd	Reserved	0	
6	RW	Start Byte Ack Interrupt Enable	0	sbyte_ack_ie
5	RW	Data Order Error Interrupt Enable	0	dat_odr_err_ie
4	RW	Bus Free Interrupt Enable	0	bus_free_ie
3	RW	Arbitration Lost Interrupt Enable	0	arb_lost_ie
2	RW	Abnormal Start/Stop Interrupt Enable	0	abn_sp_ie
1	RW	Write Data Abort Interrupt Enable	0	wdata_abt_ie
0	RW	Command Done Interrupt Enable	0	cmd_done_ie

Note: This register is used by internal processor only.

BAR0 MMIO offset – 1A07h

Register Name: I2C Interrupt Status

Size: 8 bits

Bit	Attrib	Description	Default	Mnemonic
7	Rsvd	Reserved	0	
6	RW1C	Start Byte Ack Interrupt Status	0	sbyte_ack_is
5	RW1C	Data Order Error Interrupt Status	0	dat_odr_err_is
4	RW1C	Bus Free Interrupt Status If Arbitration Lost Interrupt Status is set, master shall wait this bit is set to restart I2C operation.	0	bus_free_is
3	RW1C	Arbitration Lost Interrupt Status	0	arb_lost_is
2	RW1C	Abnormal Start/Stop Interrupt Status	0	abn_sp_is
1	RW1C	Write Data Abort Interrupt Status	0	wdata_abt_is
0	RW1C	Command Done Interrupt Status	0	cmd_done_is

Followings are sample code:

```

#define I2C_CTRL0_REG          0x1A00          //1A00h: I2C Control 0
#define I2C_CTRL1_REG          0x1A01          //1A01h: I2C Control 1
#define I2C_CLK_HIGH_REG       0x1A02          //1A02h: I2C Clock High Period
#define I2C_CLK_LOW_REG        0x1A03          //1A03h: I2C Clock Low Period
#define I2C_MASTER_DATA_REG     0x1A04          //1A04h: I2C Master R/W Data
#define I2C_MASTER_DATA_ORDER_REG 0x1A05        //1A05h: I2C Master Data Order
#define I2C_MASTER_INTE_REG     0x1A06          //1A06h: I2C Master Interrupt Enable
#define I2C_MASTER_INTS_REG     0x1A07          //1A07h: I2C Master Interrupt Status

typedef union _I2C_CTRL0 {
    struct {
        BYTE    MsRun : 1;          //bit[0]: Write 1 to trigger I2C operation in master mode. Auto-cleared when done.
        BYTE    MsTimeout : 1;      //bit[1]: Write 1 to tell HW that it's timeout. MsRun and MsTimeout will be auto-cleared.
        BYTE    Rev : 6;            //bit[7:2]: Reserved
    };
    BYTE    AsByte;
} I2C_CTRL0, *PI2C_CTRL0;

typedef union _I2C_CTRL1 {
    struct {
        BYTE    MsDriveMode : 2;     //bit[1:0]: Master drive mode
        BYTE    MsHoldTimw : 2;      //bit[3:2]: Master bus hold time control
        BYTE    Rev : 4;            //bit[7:2]: Reserved
    };
    BYTE    AsByte;
} I2C_CTRL1, *PI2C_CTRL1;

#define MASTER_STANDARD_MODE    0
#define MASTER_FAST_MODE        1
#define MASTER_FAST_PLUS__MODE 2

typedef union _I2C_MASTER_ORDER {
    struct {
        BYTE    MsOrder : 2;         //bit[1:0]: master data order
        BYTE    Rev : 6;            //bit[7:2]: Reserved
    };
    BYTE    AsByte;
} I2C_MASTER_ORDER, *PI2C_MASTER_ORDER;
  
```

```

#define MASTER_START_DATA      0
#define MASTER_DATA_ONLY      1
#define MASTER_DATA_STOP      2
#define MASTER_STOP_ONLY      3

typedef union _I2C_MASTER_INTE {
    struct {
        BYTE    MCDE : 1;          //bit[0]: Master command done interrupt enable
        BYTE    MWDAE : 1;         //bit[1]: Master Write Data Abort Interrupt Enable
        BYTE    MASSE : 1;         //bit[2]: Master Abnormal Start/Stop Interrupt Enable
        BYTE    MALE : 1;          //bit[3]: Master Arbitration Lost Interrupt Enable
        BYTE    MBFE : 1;          //bit[4]: Master Bus Free Interrupt Enable
        BYTE    MDOEE : 1;         //bit[5]: Master Data Order Error Interrupt Enable
        BYTE    MSBAE : 1;         //bit[6]: Master Start Byte Ack Interrupt Enable
        BYTE    Rev : 1;           //bit[7]: Reserved
    };
    BYTE    AsByte;
} I2C_MASTER_INTE, *PI2C_MASTER_INTE;

typedef union _I2C_MASTER_INTS {
    struct {
        BYTE    MCDS : 1;          //bit[0]: Master command done interrupt Status
        BYTE    MWDAS : 1;         //bit[1]: Master Write Data Abort Interrupt Status
        BYTE    MASSS : 1;         //bit[2]: Master Abnormal Start/Stop Interrupt Status
        BYTE    MALS : 1;          //bit[3]: Master Arbitration Lost Interrupt Status
        BYTE    MBFS : 1;          //bit[4]: Master Bus Free Interrupt Status
        BYTE    MDOES : 1;         //bit[5]: Master Data Order Error Interrupt Status
        BYTE    MSBAS : 1;         //bit[6]: Master Start Byte Ack Interrupt Status
        BYTE    Rev : 1;           //bit[7]: Reserved
    };
    BYTE    AsByte;
} I2C_MASTER_INTS, *PI2C_MASTER_INTS;

typedef enum _I2C_MS_EVENT
{
    MS_DONE = 0,                  //Master command done
    MS_ABORT,                     //Master Write Data Abort
    MS_ABNOR,                     //Master Abnormal Start/Stop
    MS_ARB_LOST,                  //Master Arbitration Lost
    MS_ORD_ERR,                   //Master Data Order Error
    MS_SB_ACK,                    //Master Start Byte Ack
    MS_TIMEOUT,                   //Master timeout
    MS_UNKNOWN = 0xFF,            //Unknown event
} I2C_MS_EVENT;

#define I2C_MASTER_TIMEOUT      1000          //1000ms timeout for I2C Master, wait least 20ms

//*****
//Procedure: I2cMsWaitCmdDone
//Description: I2C Master wait command done
//Input:      BaseAddr    - BAR0 Base Address
//Output:     I2C_MS_EVENT
  
```

```
// Note:
//*****
I2C_MS_EVENT I2cMsWaitCmdDone(DWORD BaseAddr)
{
    I2C_MS_EVENT      Result;
    BOOLEAN            Timeout;
    clock_t            ticks1, ticks2;
    I2C_MASTER_INTS    MsIs;
    I2C_CTRL0          Ctrl0;

    Result = MS_UNKNOWN;
    Timeout = FALSE;
    ticks1 = clock();
    ticks2 = ticks1;

    while( (Result == MS_UNKNOWN) && (Timeout == FALSE) )
    {
        //Read I2C Master Interrupt Status
        MsIs.AsByte = ASM116xPrivateControlReadByte(BaseAddr, I2C_MASTER_INTS_REG);
        if(MsIs.MCDS == 1)           //bit[0]: Master command done interrupt Status
        {
            Result = MS_DONE;
        }
        else if(MsIs.MWDAS == 1)     //bit[1]: Master Write Data Abort Interrupt Status
        {
            Result = MS_ABORT;
        }
        else if(MsIs.MASSS == 1)     //bit[2]: Master Abnormal Start/Stop Interrupt Status
        {
            Result = MS_ABNOR;
        }
        else if(MsIs.MALS == 1)     //bit[3]: Master Arbitration Lost Interrupt Status
        {
            Result = MS_ARB_LOST;
        }
        else if(MsIs.MDOES == 1)     //bit[5]: Master Data Order Error Interrupt Status
        {
            Result = MS_ORD_ERR;
        }
        else if(MsIs.MSBAS == 1)     //bit[6]: Master Start Byte Ack Interrupt Status
        {
            Result = MS_SB_ACK;
        }
        ticks2 = clock();
        Timeout = ( ( ((double)(ticks2 - ticks1)) / CLOCKS_PER_SEC * 1000 ) > I2C_MASTER_TIMEOUT ) ? TRUE : FALSE;
    }

    if( (Result == MS_UNKNOWN) && (Timeout == TRUE) )
    {
        Ctrl0.AsByte = 0;
        Ctrl0.MsTimeout = 1;
        //write Timeout bit
        ASM116xPrivateControlWriteByte(BaseAddr, I2C_CTRL0_REG, Ctrl0.AsByte);
    }
}
```



```

    //wait Timeout and Run bit are 0
    do
    {
        Ctrl0.AsByte = ASM116xPrivateControlReadByte(BaseAddr, I2C_CTRL0_REG);
    } while( (Ctrl0.MsTimeout == 1) || (Ctrl0.MsRun == 1) );
    Result = MS_TIMEOUT;
}

if(Result == MS_ARB_LOST)
{
    //Wait bus free
    do
    {
        MsIs.AsByte = ASM116xPrivateControlReadByte(BaseAddr, I2C_MASTER_INTS_REG);
    } while(MsIs.MBFS == 0);
}

//Clear I2C Interrupt Status register;
MsIs.AsByte = ASM116xPrivateControlReadByte(BaseAddr, I2C_MASTER_INTS_REG);
ASM116xPrivateControlWriteByte(BaseAddr, I2C_MASTER_INTS_REG, MsIs.AsByte);
//Wait run bit = 0
do
{
    Ctrl0.AsByte = ASM116xPrivateControlReadByte(BaseAddr, I2C_CTRL0_REG);
} while(Ctrl0.MsRun == 1);

return Result;
}

//*****
//Procedure: I2cMsWriteData
//Description: I2C Master Write Data
//Input:      BaseAddr  - BAR0 Base Address
//           bData      - Byte data
//           DataOrder   - Master data order, must be 0, 1, 2
//Output:     TRUE      - Success
//           FALSE      - Fail
//Note:
//*****
BOOLEAN I2cMsWriteData(DWORD BaseAddr, BYTE bData, BYTE DataOrder)
{
    I2C_MASTER_ORDER    MsOrder;
    I2C_CTRL0           Ctrl0;
    I2C_MS_EVENT         Result;

    //Write Order register
    MsOrder.AsByte = 0;
    MsOrder.MsOrder = DataOrder;
    ASM116xPrivateControlWriteByte(BaseAddr, I2C_MASTER_DATA_ORDER_REG, MsOrder.AsByte);
    //Write Master Data register
    ASM116xPrivateControlWriteByte(BaseAddr, I2C_MASTER_DATA_REG, bData);
    //Master Run
    Ctrl0.AsByte = 0;

```

```

    Ctrl0.MsRun = 1;
    ASM116xPrivateControlWriteByte(BaseAddr, I2C_CTRL0_REG, Ctrl0.AsByte);
    //Wait command done
    Result = I2cMsWaitCmdDone(BaseAddr);
    if(Result != MS_DONE)
    {
        if(Result != MS_ARB_LOST)
        {
            I2cMsStopOnly(BaseAddr);
        }
        return FALSE;
    }

    return TRUE;
}

//*****
//Procedure: I2cMsReadData
//Description: I2C Master Read Data
//Input:      BaseAddr  - BAR0 Base Address
//            pData     - Byte pointer
//            DataOrder - Master data order, must be 0, 1, 2
//Output:     TRUE      - Success
//            FALSE     - Fail
//Note:
//*****
BOOLEAN I2cMsReadData(DWORD BaseAddr, PBYTE pData, BYTE DataOrder)
{
    I2C_MASTER_ORDER  MsOrder;
    I2C_CTRL0         Ctrl0;
    I2C_MS_EVENT       Result;

    //Write Order register
    MsOrder.AsByte = 0;
    MsOrder.MsOrder = DataOrder;
    ASM116xPrivateControlWriteByte(BaseAddr, I2C_MASTER_DATA_ORDER_REG, MsOrder.AsByte);
    //Master Run
    Ctrl0.AsByte = 0;
    Ctrl0.MsRun = 1;
    ASM116xPrivateControlWriteByte(BaseAddr, I2C_CTRL0_REG, Ctrl0.AsByte);
    //Wait command done
    Result = I2cMsWaitCmdDone(pl2cControlExtension);
    //Wait command done
    Result = I2cMsWaitCmdDone(BaseAddr);
    if(Result != MS_DONE)
    {
        if(Result != MS_ARB_LOST)
        {
            I2cMsStopOnly(BaseAddr);
        }
        return FALSE;
    }
}
//Read Master Data register

```

```

        *pData = ASM116xPrivateControlReadByte(BaseAddr, I2C_MASTER_DATA_REG);

    return TRUE;
}

//*****
//Procedure: I2cMsStopOnly
//Description: I2C Master Stop only
//Input:      BaseAddr    - BAR0 Base Address
//Output:     None
//Note:
//*****
VOID I2cMsStopOnly(DWORD BaseAddr)
{
    I2C_MASTER_ORDER    MsOrder;
    I2C_CTRL0            Ctrl0;
    I2C_MS_EVENT         Result;

    //Write Order register
    MsOrder.AsByte = 0;
    MsOrder.MsOrder = MASTER_STOP_ONLY;
    ASM116xPrivateControlWriteByte(BaseAddr, I2C_MASTER_DATA_ORDER_REG, MsOrder.AsByte);
    //Master Run
    Ctrl0.AsByte = 0;
    Ctrl0.MsRun = 1;
    ASM116xPrivateControlWriteByte(BaseAddr, I2C_CTRL0_REG, Ctrl0.AsByte);
    //Wait command done
    Result = I2cMsWaitCmdDone(BaseAddr);

    return;
}

```

4.1. Write Byte Example

	7	1	1	8	1	8	1	
S	Address	Wr	A	Command Code	A	Data Byte	A	P

```

//*****
//Procedure: I2cByteWrite
//Description: I2C Byte Write
//
//          St, WsAddrW, WmAddr, WData, Sp
//
//Input:    BaseAddr    - BAR0 Base Address
//          SlvAddr      - I2C 8-bit slave address, don't care bit 0
//          MemAddr      - Command Code
//          bData         - Written byte data
//Output:    TRUE        - success
//          FALSE        - fail
//Note:
//*****

```

```

BOOLEAN I2cByteWrite(DWORD BaseAddr, BYTE SlvAddr, BYTE MemAddr, BYTE bData)
{
    BYTE    Addr;

    //Write Start + SlaveAddrW
    Addr = SlvAddr & 0xFE;           //bit 0 = 0, Write
    if(I2cMsWriteData(BaseAddr, Addr, MASTER_START_DATA) == FALSE)
    {
        return FALSE;
    }
    //Write memory address (command code)
    if(I2cMsWriteData(BaseAddr, MemAddr, MASTER_DATA_ONLY) == FALSE)
    {
        return FALSE;
    }
    //Write Data + Stop
    if(I2cMsWriteData(BaseAddr, bData, MASTER_DATA_STOP) == FALSE)
    {
        return FALSE;
    }

    return TRUE;
}

```

4.2. Read Byte Example



```

//*****
//Procedure: I2cRandomRead
//Description: I2C Random Read
//
//      St, WsAddrW, WmAddr, St, WsAddrR, RData, Sp
//
//Input:   BaseAddr  - BAR0 Base Address
//          SlvAddr   - I2C 8-bit slave address, don't care bit 0
//          MemAddr   - Command Code
//          pData     - byte data point
//Output:  TRUE      - success
//          FALSE    - fail
//Note:
//*****
BOOLEAN I2cRandomRead(DWORD BaseAddr, BYTE SlvAddr, BYTE MemAddr, PBYTE pData)
{
    BYTE    Addr;

```

```
//Write Start + SlaveAddrW
Addr = SlvAddr & 0xFE;          //bit 0 = 0, Write
if(I2cMsWriteData(BaseAddr, Addr, MASTER_START_DATA) == FALSE)
{
    return FALSE;
}
//Write memory address
if(I2cMsWriteData(BaseAddr, MemAddr, MASTER_DATA_ONLY) == FALSE)
{
    return FALSE;
}
//Write Start + SlaveAddrR
Addr = SlvAddr | 0x01;          //bit 0 = 1, Read
if(I2cMsWriteData(BaseAddr, Addr, MASTER_START_DATA) == FALSE)
{
    return FALSE;
}
//Read Data + Stop
if(I2cMsReadData(BaseAddr, pData, MASTER_DATA_STOP) == FALSE)
{
    return FALSE;
}

return TRUE;
}
```

5. GPIO Registers

There are several GPIO pins provided by ASM1064/116x series. Next table shows the provided GPIO pin numbers of different products. GPIO 8~13 and DEVSLP 0~5 are shared pins. They can be controlled by GPIO_MODE register and set DEVSLP 0~5 by default.

ASM1064/116x series GPIO Pin Numbers

Product	ASM1062A	ASM1064	ASM1164	ASM1165	ASM1166
GPIO	0, 1 8, 9	0, 1, 2, 3, 4, 5 8, 9, 10, 11	8, 9, 10, 11	0, 1, 2, 3 8, 9, 10, 11, 12	0, 1, 2, 3, 4, 5 8, 9, 10, 11, 12, 13
DEVSLP	0, 1	0, 1, 2, 3	0, 1, 2, 3	0, 1, 2, 3, 4	0, 1, 2, 3, 4, 5

Followings are registers description of GPIO pins. The bit is reserved if GPIO pin is not existing.

BAR0 MMIO offset – 1D68h

Register Name: GPIO_IN

Size: 16 bits

Bit	Attrib	Description	Default	Mnemonic
15:14	Rsvd	Reserved		
13	RW	GPIO13 input value (DEVSLP5)		
12	RW	GPIO12 input value (DEVSLP4)		
11	RW	GPIO11 input value (DEVSLP3)		
10	RW	GPIO10 input value (DEVSLP2)		
9	RW	GPIO9 input value (DEVSLP1)		
8	RW	GPIO8 input value (DEVSLP0)		
7:6	Rsvd	Reserved		
5	RW	GPIO5 input value		
4	RW	GPIO4 input value		
3	RW	GPIO3 input value		
2	RW	GPIO2 input value		
1	RW	GPIO1 input value		
0	RW	GPIO0 input value		

Note: When GPIO13 ~ 8 are set DEVSLP, bit13 ~8 are reserved

BAR0 MMIO offset – 1D6Ah

Register Name: GPIO_CTL

Size: 16 bits

Bit	Attrib	Description	Default	Mnemonic
15:14	Rsvd	Reserved	0	
13	RW	GPIO13 output enable (DEVSLP5)	0	
12	RW	GPIO12 output enable (DEVSLP4)	0	
11	RW	GPIO11 output enable (DEVSLP3)	0	
10	RW	GPIO10 output enable (DEVSLP2)	0	
9	RW	GPIO9 output enable (DEVSLP1)	0	

8	RW	GPIO8 output enable (DEVSLP0)	0	
7:6	Rsvd	Reserved	0	
5	RW	GPIO5 output enable	0	
4	RW	GPIO4 output enable	0	
3	RW	GPIO3 output enable	0	
2	RW	GPIO2 output enable	0	
1	RW	GPIO1 output enable	0	
0	RW	GPIO0 output enable	0	

Note: When GPIO13 ~ 8 are set DEVSLP, bit13 ~8 are reserved

BAR0 MMIO offset – 1D6Ch

Register Name: GPIO_OUT

Size: 16 bits

Bit	Attrib	Description	Default	Mnemonic
15:14	Rsvd	Reserved	0	
13	RW	GPIO13 ouput value (DEVSLP5)	0	
12	RW	GPIO12 ouput value (DEVSLP4)	0	
11	RW	GPIO11 ouput value (DEVSLP3)	0	
10	RW	GPIO10 ouput value (DEVSLP2)	0	
9	RW	GPIO9 ouput value (DEVSLP1)	0	
8	RW	GPIO8 ouput value (DEVSLP0)	0	
7:6	Rsvd	Reserved	0	
5	RW	GPIO5 ouput value	0	
4	RW	GPIO4 ouput value	0	
3	RW	GPIO3 ouput value	0	
2	RW	GPIO2 ouput value	0	
1	RW	GPIO1 ouput value	0	
0	RW	GPIO0 ouput value	0	

Note: When GPIO13 ~ 8 are set DEVSLP, bit13 ~8 are reserved

BAR0 MMIO offset – 1D6Eh

Register Name: GPIO_MODE

Size: 8 bits

Bit	Attrib	Description	Default	Mnemonic
7:6	Rsvd	Reserved	0	
5	RW	GPIO13 Mode 0: DEVSLP5 1: GPIO13	0	
4	RW	GPIO12 Mode 0: DEVSLP4 1: GPIO12	0	
3	RW	GPIO11 Mode 0: DEVSLP3 1: GPIO11	0	
2	RW	GPIO10 Mode 0: DEVSLP2 1: GPIO10	0	
1	RW	GPIO9 Mode	0	

		0: DEVSLP1 1: GPIO9		
0	RW	GPIO8 Mode 0: DEVSLP0 1: GPIO8	0	

Followings are sample code:

```
#define GPIO_INPUT_REG          0x1D68          //1D68h: GPIO input value
#define GPIO_CTRL_REG          0x1D6A          //1D6Ah: GPIO input/output control
#define GPIO_OUTPUT_REG        0x1D6C          //1D6Ch: GPIO output value
#define GPIO_MODE_REG          0x1D6E          //1D6Eh: GPIO 8~13 mode control
```

```
/******
```

```
//Procedure: GetGpioInputValue
```

```
//Description: Get a GPIO pin input value
```

```
//Input:      BaseAddr          - BAR0 Base Address
```

```
//           GpioNumber        - GPIO pin number
```

```
//Output:     input value (0 or 1)
```

```
// Note:
```

```
/******
```

```
UINT GetGpioInputValue(DWORD BaseAddr, UINT GpioNumber)
```

```
{
    WORD    GpioValues;

    GpioValues = ASM116xPrivateControlReadWord(BaseAddr, GPIO_INPUT_REG);
    GpioValues = GpioValues & (1 << GpioNumber);
    if(GpioValues != 0)
    {
        return 1;
    }
    else
    {
        return 0;
    }
}
```

```
/******
```

```
//Procedure: EnableGpioOutput
```

```
//Description: Enable a GPIO pin to output
```

```
//Input:      BaseAddr          - BAR0 Base Address
```

```
//           GpioNumber        - GPIO pin number
```

```
//Output:     None
```

```
//Note:
```

```
/******
```

```
VOID EnableGpioOutput(DWORD BaseAddr, UINT GpioNumber)
```

```
{
    WORD    GpioCtrl;

    GpioCtrl = ASM116xPrivateControlReadWord(BaseAddr, GPIO_CTRL_REG);
    GpioCtrl = GpioCtrl | (1 << GpioNumber);
    ASM116xPrivateControlWriteWord(BaseAddr, GPIO_CTRL_REG, GpioCtrl);
}
```



```
        return;
    }

    /*******
    //Procedure: SetGpioOutputValue
    //Description: Set a GPIO pin output value
    //Input:      BaseAddr      - BAR0 Base Address
    //            GpioNumber    - GPIO pin number
    //            OutputValue   - 0 or 1
    //Output:     None
    //Note:
    /*******
    VOID SetGpioOutputValue(DWORD BaseAddr, UINT GpioNumber, UINT OutputValue)
    {
        WORD        GpioValues;

        GpioValues = ASM116xPrivateControlReadWord(BaseAddr, GPIO_OUTPUT_REG);
        if(OutputValue == 0)
        {
            GpioValues = GpioValues & ~(1 << GpioNumber);
        }
        else
        {
            GpioValues = GpioValues | (1 << GpioNumber);
        }
        ASM116xPrivateControlWriteWord(BaseAddr, GPIO_OUTPUT_REG, GpioValues);

        return;
    }
```

6. LED Registers

There are several LED pins provided by ASM1064/116x series. Next table shows the provided LED pin numbers of different products.

ASM116x series LED Pin Numbers

Product	ASM1062A	ASM1064	ASM1164	ASM1165	ASM1166
LED	0, 1	0, 1, 2, 3	0, 1, 2, 3	0, 1, 2, 3, 4	0, 1, 2, 3, 4, 5

There are two state of LED – Idle, and Busy state. Each state can be programmed individually. Next table shows the behaviors at Idle and Busy state.

ASM1064/116x series LED Behaviors

	ON	OFF	Blink	Breathe
Idle State	V	V	V	X
Busy State	V	V	V	V

Idle State: SATA port has no command in process

Busy State: SATA port has command in process or data is transferring

Followings are registers description of LED pins. The register or bit is reserved if LED pin is not exist.

BAR0 MMIO offset – 1D00h, 1D01h, (ASM1062A/1064/1164/1165/1166 LED 0, 1)
1D02h, 1D03h, (ASM1064/1164/1165/1166 LED 2, 3)
1D04h, (ASM1165/1166 LED 4)
1D05h (ASM1166 LED 5)

Register Name: LED_OFF_P0/1/2/3/4/5

Size: 8 bits

Bit	Attrib	Description	Default	Mnemonic
7:0	RW	Port n LED off duration in idle state Duration = value * Port n LED time unit FFh: always off	FFh	

BAR0 MMIO offset – 1D08h, 1D09h, (ASM1062A/1064/1164/1165/1166 LED 0, 1)
1D0Ah, 1D0Bh, (ASM1064/1164/1165/1166 LED 2, 3)
1D0Ch, (ASM1165/1166 LED 4)
1D0Dh (ASM1166 LED 5)

Register Name: LED_ON_P0/1/2/3/4/5

Size: 8 bits

Bit	Attrib	Description	Default	Mnemonic
7:0	RW	Port n LED on duration in idle state Duration = value * Port n LED time unit FFh: always on	0	

BAR0 MMIO offset – 1D10h, 1D11h, (ASM1062A/1064/1164/1165/1166 LED 0, 1)
1D12h, 1D13h, (ASM1064/1164/1165/1166 LED 2, 3)
1D14h, (ASM1165/1166 LED 4)
1D15h (ASM1166 LED 5)

Register Name: ALED_OFF_P0/1/2/3/4/5

Size: 8 bits

Bit	Attrib	Description	Default	Mnemonic
7:0	RW	Port n LED off duration in busy state at normal mode Duration = value * Port n LED time unit FFh: always off	0	

BAR0 MMIO offset – 1D18h, 1D19h, (ASM1062A/1064/1164/1165/1166 LED 0, 1)
1D1Ah, 1D1Bh, (ASM1064/1164/1165/1166 LED 2, 3)
1D1Ch, (ASM1165/1166 LED 4)
1D1Dh (ASM1166 LED 5)

Register Name: ALED_ON_P0/1/2/3/4/5

Size: 8 bits

Bit	Attrib	Description	Default	Mnemonic
7:0	RW	Port n LED on duration in busy state at normal mode Duration = value * Port n LED time unit FFh: always off	FFh	

BAR0 MMIO offset – 1D20h

Register Name: LED_UNIT

Size: 16 bits

Bit	Attrib	Description	Default	Mnemonic
15:12	Rsvd	Reserved	0	
11:10	RW	Port 5 LED time unit 0: 100 us, 1: 1 ms, 2: 10 ms, 3: 100 ms	2	
9:8	RW	Port 4 LED time unit 0: 100 us, 1: 1 ms, 2: 10 ms, 3: 100 ms	2	
7:6	RW	Port 3 LED time unit 0: 100 us, 1: 1 ms, 2: 10 ms, 3: 100 ms	2	
5:4	RW	Port 2 LED time unit 0: 100 us, 1: 1 ms, 2: 10 ms, 3: 100 ms	2	
3:2	RW	Port 1 LED time unit 0: 100 us, 1: 1 ms, 2: 10 ms, 3: 100 ms	2	
1:0	RW	Port 0 LED time unit 0: 100 us, 1: 1 ms, 2: 10 ms, 3: 100 ms	2	

BAR0 MMIO offset – 1D22h

Register Name: LED_EN

Size: 8 bits

Bit	Attrib	Description	Default	Mnemonic
7:6	Rsvd	Reserved	0	
5	RW	Port 5 LED enable 0: Disabled, 1: Enabled	1	
4	RW	Port 4 LED enable 0: Disabled, 1: Enabled	1	
3	RW	Port 3 LED enable 0: Disabled, 1: Enabled	1	
2	RW	Port 2 LED enable 0: Disabled, 1: Enabled	1	
1	RW	Port 1 LED enable 0: Disabled, 1: Enabled	1	
0	RW	Port 0 LED enable 0: Disabled, 1: Enabled	1	

Note: Don't disable if LEDs are used

BAR0 MMIO offset – 1D23h

Register Name: ALED_EN

Size: 8 bits

Bit	Attrib	Description	Default	Mnemonic
7:6	Rsvd	Reserved	0	
5	RW	Port 5 Active LED normal mode enable 0: Disabled, 1: Enabled	1	
4	RW	Port 4 Active LED normal mode enable 0: Disabled, 1: Enabled	1	
3	RW	Port 3 Active LED normal mode enable 0: Disabled, 1: Enabled	1	
2	RW	Port 2 Active LED normal mode enable 0: Disabled, 1: Enabled	1	
1	RW	Port 1 Active LED normal mode enable 0: Disabled, 1: Enabled	1	
0	RW	Port 0 Active LED normal mode enable 0: Disabled, 1: Enabled	1	

Note: Don't disable ALED normal mode if LEDs are used

BAR0 MMIO offset – 1D24h

Register Name: LED_STATE

Size: 8 bits

Bit	Attrib	Description	Default	Mnemonic
7:6	Rsvd	Reserved	0	
5	RW	Port 5 LED State Control 0: LED in busy/idle state is controlled by HW 1: Force LED in busy state	0	
4	RW	Port 4 LED State Control 0: LED in busy/idle state is controlled by HW 1: Force LED in busy state	0	
3	RW	Port 3 LED State Control 0: LED in busy/idle state is controlled by HW 1: Force LED in busy state	0	
2	RW	Port 2 LED State Control 0: LED in busy/idle state is controlled by HW 1: Force LED in busy state	0	

1	RW	Port 1 LED State Control 0: LED in busy/idle state is controlled by HW 1: Force LED in busy state	0	
0	RW	Port 0 LED State Control 0: LED in busy/idle state is controlled by HW 1: Force LED in busy state	0	

BAR0 MMIO offset – 1D25h

Register Name: LED_OP_MODE

Size: 8 bits

Bit	Attrib	Description	Default	Mnemonic
7:1	Rsvd	Reserved	0	
0	RW	LED operation mode 0: Low active 1: High active	0	

BAR0 MMIO offset – 1D30h

Register Name: ALED_BM_OFF

Size: 8 bits

Bit	Attrib	Description	Default	Mnemonic
7:0	RW	Active LED initial off duration in busy state at Breath Mode Duration = value * Active LED Breath mode time unit	0Ah	

BAR0 MMIO offset – 1D31h

Register Name: ALED_BM_ON

Size: 8 bits

Bit	Attrib	Description	Default	Mnemonic
7:0	RW	Active LED initial on duration in busy state at Breath Mode Duration = value * Active LED Breath mode time unit	5Ah	

BAR0 MMIO offset – 1D32h

Register Name: ALED_BM_ILOOP_CNT

Size: 8 bits

Bit	Attrib	Description	Default	Mnemonic
7:0	RW	Active LED inner loop count in busy state at Breath Mode The duration will not be changed during this loop	05h	

BAR0 MMIO offset – 1D33h

Register Name: ALED_BM_OLOOP_CNT

Size: 8 bits

Bit	Attrib	Description	Default	Mnemonic
7:0	RW	Active LED outer loop count in busy state at Breath Mode The duration will be changed during this loop	1Eh	

BAR0 MMIO offset – 1D34h

Register Name: ALED_BM_STEP_SIZE

Size: 8 bits

Bit	Attrib	Description	Default	Mnemonic
7:0	RW	Active LED step size in busy state at Breath Mode The duration will be changed during this loop	03h	

BAR0 MMIO offset – 1D35h

Register Name: ALED_BM_CTL

Size: 8 bits

Bit	Attrib	Description	Default	Mnemonic
7:4	Rsvd	Reserved	0	
3:2	RW	Active LED Breath mode time unit 0: 10 us, 1: 100 us, 2: 1 ms, 3: 10 ms	1	
1	RW	Active LED Breath mode enable 0: Disabled, 1: Enabled	1	
0	RW	Active LED enable 0: Disabled, 1: Enabled	1	

Note1: Don't modify bit[1:0] if LEDs are used

Note2: Breath mode algorithm:

```

LED_ON = LED_INIT_ON;
FOREVER {
    FOR (MODE=0; MODE<2; MODE=MODE+1) {
        FOR (OLOOP=0; OLOOP<OLOOP_CNT; OLOOP=OLOOP+1) {
            FOR (ILOOP=0; ILOOP<ILOOP_CNT; ILOOP=ILOOP+1) {}
            IF (MODE=0) {
                LED_ON = LED_ON - STEP_SIZE;
            } else {
                LED_ON = LED_ON + STEP_SIZE;
            }
        }
    }
}

```

BAR0 MMIO offset – 1D38h

Register Name: LED_IO_MODE

Size: 16 bits

Bit	Attrib	Description	Default	Mnemonic
15:12	Rsvd	Reserved	0	
11:10	RW	LED5 I/O Mode 0: Normal mode active LED	0	

		1: Breath mode active LED 2: Reserved 3: GPIO		
9:8	RW	LED4 I/O Mode 0: Normal mode active LED 1: Breath mode active LED 2: Reserved 3: GPIO	0	
7:6	RW	LED3 I/O Mode 0: Normal mode active LED 1: Breath mode active LED 2: Reserved 3: GPIO	0	
5:4	RW	LED2 I/O Mode 0: Normal mode active LED 1: Breath mode active LED 2: Reserved 3: GPIO	0	
3:2	RW	LED1 I/O Mode 0: Normal mode active LED 1: Breath mode active LED 2: Reserved 3: GPIO	0	
1:0	RW	LED0 I/O Mode 0: Normal mode active LED 1: Breath mode active LED 2: Reserved 3: GPIO	0	

BAR0 MMIO offset – 1D3Ah

Register Name: LED_GPO

Size: 8 bits

Bit	Attrib	Description	Default	Mnemonic
7:6	Rsvd	Reserved	0	
5	RW	LED5 GPO output enable 0: Disabled, 1: Enabled	0	
4	RW	LED4 GPO output enable 0: Disabled, 1: Enabled	0	
3	RW	LED3 GPO output enable 0: Disabled, 1: Enabled	0	
2	RW	LED2 GPO output enable 0: Disabled, 1: Enabled	0	
1	RW	LED1 GPO output enable 0: Disabled, 1: Enabled	0	
0	RW	LED0 GPO output enable 0: Disabled, 1: Enabled	0	

Note 1: This register is used when LED I/O Mode is set GPIO.

Note 2: The output value depends on “LED operation mode” bit

BAR0 MMIO offset – 1D3Bh

Register Name: LED_GPI

Size: 8 bits

Bit	Attrib	Description	Default	Mnemonic
-----	--------	-------------	---------	----------

7:6	Rsvd	Reserved	0	
5	RO	LED5 GPI Value		
4	RO	LED4 GPI Value		
3	RO	LED3 GPI Value		
2	RO	LED2 GPI Value		
1	RO	LED1 GPI Value		
0	RO	LED0 GPI Value		

Followings are sample code:

```

#define ALED0_IDLE_OFF_REG      0x1D00    //ALED0 off duration in idle state
#define ALED1_IDLE_OFF_REG      0x1D01    //ALED1 off duration in idle state
#define ALED2_IDLE_OFF_REG      0x1D02    //ALED2 off duration in idle state
#define ALED3_IDLE_OFF_REG      0x1D03    //ALED3 off duration in idle state
#define ALED4_IDLE_OFF_REG      0x1D04    //ALED4 off duration in idle state
#define ALED5_IDLE_OFF_REG      0x1D05    //ALED5 off duration in idle state
#define ALED0_IDLE_ON_REG       0x1D08    //ALED0 on duration in idle state
#define ALED1_IDLE_ON_REG       0x1D09    //ALED1 on duration in idle state
#define ALED2_IDLE_ON_REG       0x1D0A    //ALED2 on duration in idle state
#define ALED3_IDLE_ON_REG       0x1D0B    //ALED3 on duration in idle state
#define ALED4_IDLE_ON_REG       0x1D0C    //ALED4 on duration in idle state
#define ALED5_IDLE_ON_REG       0x1D0D    //ALED5 on duration in idle state
#define ALED0_BUSY_OFF_REG      0x1D10    //ALED0 off duration in busy state at normal mode
#define ALED1_BUSY_OFF_REG      0x1D11    //ALED1 off duration in busy state at normal mode
#define ALED2_BUSY_OFF_REG      0x1D12    //ALED2 off duration in busy state at normal mode
#define ALED3_BUSY_OFF_REG      0x1D13    //ALED3 off duration in busy state at normal mode
#define ALED4_BUSY_OFF_REG      0x1D14    //ALED4 off duration in busy state at normal mode
#define ALED5_BUSY_OFF_REG      0x1D15    //ALED5 off duration in busy state at normal mode
#define ALED0_BUSY_ON_REG       0x1D18    //ALED0 on duration in busy state at normal mode
#define ALED1_BUSY_ON_REG       0x1D19    //ALED1 on duration in busy state at normal mode
#define ALED2_BUSY_ON_REG       0x1D1A    //ALED2 on duration in busy state at normal mode
#define ALED3_BUSY_ON_REG       0x1D1B    //ALED3 on duration in busy state at normal mode
#define ALED4_BUSY_ON_REG       0x1D1C    //ALED4 on duration in busy state at normal mode
#define ALED5_BUSY_ON_REG       0x1D1D    //ALED5 on duration in busy state at normal mode
#define ALED_TIME_UINT_REG      0x1D20    //ALED0 ~ 5 time unit for idle state and busy state at normal mode
#define ALED_ENABLE_REG         0x1D22    //ALED0 ~ 5 enable
#define ALED_NORMAL_ENABLE_REG  0x1D23    //ALED0 ~ 5 enable Normal mode in busy state
#define ALED_STATE_CONTROL_REG  0x1D24    //ALED0 ~ 5 state control
#define ALED_OP_MODE_REG        0x1D25    //ALED operation mode control
#define ALED_BREATH_INNER_OFF_REG 0x1D30    //ALED breathing off duration in inner loop of busy state at breathing mode
#define ALED_BREATH_INNER_ON_REG 0x1D31    //ALED breathing on duration in inner loop of busy state at breathing mode
#define ALED_BREATH_INNER_COUNT_REG 0x1D32    //ALED breathing inner loop count of busy state at breathing mode
#define ALED_BREATH_OUTER_COUNT_REG 0x1D33    //ALED breathing outer loop count of busy state at breathing mode
#define ALED_BREATH_STEP_SIZE_REG 0x1D34    //ALED step size in inner loop of busy state at breathing mode
#define ALED_BREATH_CONTROL_REG 0x1D35    //ALED breathing control
#define ALED_IO_MODE_REG        0x1D38    //ALED IO mode control
#define ALED_GPO_ENABLE_REG      0x1D3A    //ALED0 ~ 5 GPO enable
#define ALED_GPI_VALUE_REG       0x1D3B    //ALED0 ~ 5 GPI value

```

```

typedef union _ALED_TIME_UNIT {
    struct {

```



```

    WORD    Aled0TimeUnit : 2;        //bit[1:0]: ALED0 time unit
    WORD    Aled1TimeUnit : 2;        //bit[3:2]: ALED1 time unit
    WORD    Aled2TimeUnit : 2;        //bit[5:4]: ALED2 time unit
    WORD    Aled3TimeUnit : 2;        //bit[7:6]: ALED3 time unit
    WORD    Aled4TimeUnit : 2;        //bit[9:8]: ALED4 time unit
    WORD    Aled5TimeUnit : 2;        //bit[11:10]: ALED5 time unit
    WORD    Rev : 4;                  //bit[15:12]: Reserved
};
    WORD    AsWord;
} ALED_TIME_UNIT, *PALED_TIME_UNIT;

#define ALED_TIME_UINT_100us    0        //100 us
#define ALED_TIME_UINT_1ms      1        //1 ms
#define ALED_TIME_UINT_10ms     2        //10 ms
#define ALED_TIME_UINT_100ms    3        //100 ms

typedef union _ALED_BREATH_CTRL {
    struct {
        BYTE    AledEnable : 1;        //bit[0]: ALED enable, set to 1
        BYTE    BreathEnable : 1;      //bit[1]: Breathing Enable
        BYTE    Timeunit : 2;          //bit[3:2]: Breathing on/off time unit in inner loop
        BYTE    Rev4 : 4;              //bit[7:4]: Reserved
    };
    BYTE    AsByte;
} ALED_BREATH_CTRL, *PALED_BREATH_CTRL;

#define BREATH_TIME_UINT_10us    0        //10 us
#define BREATH_TIME_UINT_100us  1        //100 us
#define BREATH_TIME_UINT_1ms     2        //1 ms
#define BREATH_TIME_UINT_10ms    3        //10 ms

typedef union _ALED_IO_CTRL {
    struct {
        WORD    Aled0IoCtrl : 2;        //bit[1:0]: ALED0 IO control
        WORD    Aled1IoCtrl : 2;        //bit[3:2]: ALED1 IO control
        WORD    Aled2IoCtrl : 2;        //bit[5:4]: ALED2 IO control
        WORD    Aled3IoCtrl : 2;        //bit[7:6]: ALED3 IO control
        WORD    Aled4IoCtrl : 2;        //bit[9:8]: ALED4 IO control
        WORD    Aled5IoCtrl : 2;        //bit[11:10]: ALED5 IO control
        WORD    Rev : 4;                //bit[15:12]: Reserved
    };
    WORD    AsWord;
} ALED_IO_CTRL, *PALED_IO_CTRL;

#define ALED_IO_NORMAL_MODE    0
#define ALED_IO_BREATH_MODE    1
#define ALED_IO_GPIO_MODE      3

//*****
//Procedure: SetAledIdleOff
//Description: Set ALED Idle State OFF
//Input:      BaseAddr        - BAR0 Base Address
//           AledNumber        - ALED number, must 0, 1, 2, 3, 4, 5

```

```

//Output:    None
//Note:
//*****
VOID SetAledIdleOff(DWORD BaseAddr, UINT AledNumber)
{
    ASM116xPrivateControlWriteByte(BaseAddr, (ALED0_IDLE_OFF_REG + AledNumber), 0xFF);
    ASM116xPrivateControlWriteByte(BaseAddr, (ALED0_IDLE_ON_REG + AledNumber), 0x00);

    return;
}

//*****
//Procedure: SetAledIdleOn
//Description: Set ALED Idle State ON
//Input:      BaseAddr      - BAR0 Base Address
//            AledNumber    - ALED number, must 0, 1, 2, 3, 4, 5
//Output:     None
//Note:
//*****
VOID SetAledIdleOn(DWORD BaseAddr, UINT AledNumber)
{
    ASM116xPrivateControlWriteByte(BaseAddr, (ALED0_IDLE_OFF_REG + AledNumber), 0x00);
    ASM116xPrivateControlWriteByte(BaseAddr, (ALED0_IDLE_ON_REG + AledNumber), 0xFF);

    return;
}

//*****
//Procedure: SetAledIdleBlink
//Description: Set ALED Idle blink 5Hz
//Input:      BaseAddr      - BAR0 Base Address
//            AledNumber    - ALED number, must 0, 1, 2, 3, 4, 5
//Output:     None
//Note:
//            BlinkOffTime and BlinkOnTime shall be greater than 100 ms for human's persistence of vision
//*****
VOID SetAledIdleBlink(DWORD BaseAddr, UINT AledNumber)
{
    ALED_TIME_UNIT AledTimeunit;
    UINT           Timeunit;

    Timeunit = ALED_TIME_UINT_1ms;
    AledTimeunit.AsWord = ASM116xPrivateControlReadWord(BaseAddr, ALED_TIME_UINT_REG);
    switch(AledNumber)
    {
        case 0:
            AledTimeunit.Aled0TimeUnit = Timeunit;
            break;
        case 1:
            AledTimeunit.Aled1TimeUnit = Timeunit;
            break;
        case 2:
            AledTimeunit.Aled2TimeUnit = Timeunit;

```

```

        break;
    case 3:
        AledTimeunit.Aled3TimeUnit = Timeunit;
        break;
    case 4:
        AledTimeunit.Aled4TimeUnit = Timeunit;
        break;
    case 5:
        AledTimeunit.Aled5TimeUnit = Timeunit;
        break;
    default:
        break;
}
//Set time unit
ASM116xPrivateControlWriteWord(BaseAddr, ALED_TIME_UINT_REG, AledTimeunit.AsWord);
//Set Off/On duration, 100 * 1ms = 100ms
ASM116xPrivateControlWriteByte(BaseAddr, (ALED0_IDLE_OFF_REG + AledNumber), 100);
ASM116xPrivateControlWriteByte(BaseAddr, (ALED0_IDLE_ON_REG + AledNumber), 100);

return;
}

//*****
//Procedure: SetAledBusyNormalOff
//Description: Set ALED OFF in busy state at normal mode
//Input:      BaseAddr      - BAR0 Base Address
//           AledNumber      - ALED number, must 0, 1, 2, 3, 4, 5
//Output:     None
//Note:
//*****
VOID SetAledBusyNormalOff(DWORD BaseAddr, UINT AledNumber)
{
    ASM116xPrivateControlWriteByte(BaseAddr, (ALED0_BUSY_OFF_REG + AledNumber), 0xFF);
    ASM116xPrivateControlWriteByte(BaseAddr, (ALED0_BUSY_ON_REG + AledNumber), 0x00);

    return;
}

//*****
//Procedure: SetAledBusyNormalOn
//Description: Set ALED OFF in busy state at normal mode
//Input:      BaseAddr      - BAR0 Base Address
//           AledNumber      - ALED number, must 0, 1, 2, 3, 4, 5
//Output:     None
//Note:
//*****
VOID SetAledBusyNormalOn(DWORD BaseAddr, UINT AledNumber)
{
    ASM116xPrivateControlWriteByte(BaseAddr, (ALED0_BUSY_OFF_REG + AledNumber), 0x00);
    ASM116xPrivateControlWriteByte(BaseAddr, (ALED0_BUSY_ON_REG + AledNumber), 0xFF);

    return;
}

```

```

//*****
//Procedure: SetAledBusyNormalBlink
//Description: Set ALED blink 5Hz in busy state at normal mode
//Input:      BaseAddr      - BAR0 Base Address
//            AledNumber     - ALED number, must 0, 1, 2, 3, 4, 5
//Output:      None
//Note:
//            BlinkOffTime and BlinkOnTime shall be greater than 100 ms for human's persistence of vision
//*****
VOID SetAledBusyNormalBlink(DWORD BaseAddr, UINT AledNumber)
{
    ALED_TIME_UNIT AledTimeunit;
    ALED_IO_CTRL   AledIoCtrl;
    UINT           Timeunit;

    Timeunit = ALED_TIME_UNIT_1ms;
    AledIoCtrl.AsWord = ASM116xPrivateControlReadWord(BaseAddr, ALED_IO_MODE_REG);
    AledTimeunit.AsWord = ASM116xPrivateControlReadWord(BaseAddr, ALED_TIME_UINT_REG);
    switch(AledNumber)
    {
        case 0:
            AledTimeunit.Aled0TimeUnit = Timeunit;
            AledIoCtrl.Aled0IoCtrl = ALED_IO_NORMAL_MODE;
            break;
        case 1:
            AledTimeunit.Aled1TimeUnit = Timeunit;
            AledIoCtrl.Aled1IoCtrl = ALED_IO_NORMAL_MODE;
            break;
        case 2:
            AledTimeunit.Aled2TimeUnit = Timeunit;
            AledIoCtrl.Aled2IoCtrl = ALED_IO_NORMAL_MODE;
            break;
        case 3:
            AledTimeunit.Aled3TimeUnit = Timeunit;
            AledIoCtrl.Aled3IoCtrl = ALED_IO_NORMAL_MODE;
            break;
        case 4:
            AledTimeunit.Aled4TimeUnit = Timeunit;
            AledIoCtrl.Aled4IoCtrl = ALED_IO_NORMAL_MODE;
            break;
        case 5:
            AledTimeunit.Aled5TimeUnit = Timeunit;
            AledIoCtrl.Aled5IoCtrl = ALED_IO_NORMAL_MODE;
            break;
        default:
            break;
    }
    //Set time unit
    ASM116xPrivateControlWriteWord(BaseAddr, ALED_TIME_UINT_REG, AledTimeunit.AsWord);
    //Set Off/On duration, 100 * 1ms = 100ms
    ASM116xPrivateControlWriteByte(BaseAddr, (ALED0_BUSY_OFF_REG + AledNumber), 100);
    ASM116xPrivateControlWriteByte(BaseAddr, (ALED0_BUSY_ON_REG + AledNumber), 100);
}

```

```

//Set busy normal mode
ASM116xPrivateControlWriteWord(BaseAddr, ALED_IO_MODE_REG, AledIoCtrl.AsWord);

return;
}

//*****
//Procedure: SetAledBusyBreath
//Description: Set ALED breathing 3 seconds in busy state
//Input:      BaseAddr      - BAR0 Base Address
//           AledNumber      - ALED number, must 0, 1, 2, 3, 4, 5
//Output:      None
//Note:
//*****
VOID SetAledBusyBreath(DWORD BaseAddr, UINT AledNumber)
{
    ALED_BREATH_CTRL    AledBreathCtrl;
    ALED_IO_CTRL         AledIoCtrl;
    UINT                 Timeunit;

    //Breathing Cycle: 3 seconds
    //OFF_DURATION = 10, ON_DURATION = 90, ILOOP_CNT = 5, OLOOP_CNT = 30, STEP_SIZE = 3
    //TIME_UNIT = 100 us
    //Breathing_cycle_time = (10 + 90) * 5 * 30 * 100 us * 2 = 3000000 us = 3000 ms = 3 seconds

    ASM116xPrivateControlWriteByte(BaseAddr, ALED_BREATH_INNER_OFF_REG, 10);
    ASM116xPrivateControlWriteByte(BaseAddr, ALED_BREATH_INNER_ON_REG, 90);
    ASM116xPrivateControlWriteByte(BaseAddr, ALED_BREATH_INNER_COUNT_REG, 5);
    ASM116xPrivateControlWriteByte(BaseAddr, ALED_BREATH_OUTER_COUNT_REG, 30);

    AledBreathCtrl.AsByte = ASM116xPrivateControlReadByte(BaseAddr, ALED_BREATH_CONTROL_REG);
    AledBreathCtrl.Timeunit = BREATH_TIME_UINT_100us;
    AledBreathCtrl.BreathEnable = 1;
    AledBreathCtrl.AledEnable = 1;
    ASM116xPrivateControlWriteByte(BaseAddr, ALED_BREATH_CONTROL_REG, AledBreathCtrl.AsByte);

    AledIoCtrl.AsWord = ASM116xPrivateControlReadWord(BaseAddr, ALED_IO_MODE_REG);
    switch(AledNumber)
    {
        case 0:
            AledIoCtrl.Aled0IoCtrl = ALED_IO_BREATH_MODE;
            break;
        case 1:
            AledIoCtrl.Aled1IoCtrl = ALED_IO_BREATH_MODE;
            break;
        case 2:
            AledIoCtrl.Aled2IoCtrl = ALED_IO_BREATH_MODE;
            break;
        case 3:
            AledIoCtrl.Aled3IoCtrl = ALED_IO_BREATH_MODE;
            break;
        case 4:
            AledIoCtrl.Aled4IoCtrl = ALED_IO_BREATH_MODE;

```

```
        break;
    case 5:
        AledIoCtrl.Aled5IoCtrl = ALED_IO_BREATH_MODE;
        break;
    default:
        break;
}
//Set busy breath mode
ASM116xPrivateControlWriteWord(BaseAddr, ALED_IO_MODE_REG, AledIoCtrl.AsWord);

return;
}
```

ASMedia Confidential

7. PCI Express Capability Registers

The PCI Express Capability Structure starts from PCI CFG offset 80h. ASM1064/116x supports - PCI Express Capability List, PCI Express Capabilities, Device Capabilities/Control/Status, Link Capabilities/Control/Status, and Link Capabilities 2 Register. Advanced Error Reporting Capability Registers are supported in PCI CFG offset 100h.

PCI CFG
Offset

	31	24	23	16	15	8	7	0
80h	PCI Express Capabilities Register				Next Capability Pointer		Capability ID	
84h	Device Capabilities Register							
88h	Device Status Register				Device Control Register			
8Ch	Link Capabilities Register							
90h	Link Status Register				Link Control Register			
94h	Reserved for ASM1064/116x							
98h								
9Ch								
A0h								
A4h								
A8h	Link Capabilities 2 Register							
ACH								
B0h								
...								
...								
FCh	Advanced Error Reporting Capability Structure							
100h								
...								
...								
128h								

7.1. PCI Express Capability List Register

PCI CFG offset – 80h

Register Name: PCIE_CAPL

Size: 16 bits

Bit	Attrib	Description	Default	Mnemonic
15:8	RO	Next Capability Pointer	00h	
7:0	RO	Capability ID Indicates the PCI Express Capability structure.	10h	

7.2. PCI Express Capabilities Register

PCI CFG offset – 82h

Register Name: PCIE_CAP

Size: 16 bits

Bit	Attrib	Description	Default	Mnemonic
15	Rsvd	Reserved	0	
14	RO	Undefined	0	
13:9	RO	Interrupt Message Number	00h	
8	RO	Slot Implemented	0	
7:4	RO	Device/Port Type 0000b PCI Express Endpoint	0h	
3:0	RO	Capability Version	2h	

7.3. Device Capabilities Register

PCI CFG offset – 84h

Register Name: DEV_CAP

Size: 32 bits

Bit	Attrib	Description	Default	Mnemonic
31:29	Rsvd	Reserved	0h	
28	RO	Function Level Reset Capability	0	
27:26	RO	Captured Slot Power Limit Scale	0	
25:18	RO	Captured Slot Power Limit Value	00h	
15	RO	Role-Based Error Reporting	1	
14:12	RO	Undefined	0	
11:9	RO	Endpoint L1 Acceptable Latency 000b Maximum of 1 μ s	0h	
8:6	RO	Endpoint L0s Acceptable Latency 000b Maximum of 64 ns	0h	
5	RO	Extended Tag Field Supported 0b 5-bit Tag field supported 1b 8-bit Tag field supported	1	
4:3	RO	Phantom Functions Supported 00b No Function Number bits are used for Phantom Functions.	0	
2:0	RO	Max_Payload_Size Supported 010b 512 bytes max payload size	2h	

7.4. Device Control Register

PCI CFG offset – 88h

Register Name: DEV_CTRL

Size: 16 bits

Bit	Attrib	Description	Default	Mnemonic
15	Rsvd	Reserved	0	
14:12	RW	Max Read Request Size 000b 128 bytes maximum Read Request size 001b 256 bytes maximum Read Request size 010b 512 bytes maximum Read Request size others Reserved for ASM1064/116x	2h	
11	RW	Enable No Snoop	1	
10	RW	Aux Power PM Enable	0	
9	RW	Phantom Functions Enable	0	
8	RW	Extended Tag Field Enable	0	
7:5	RW	Max Payload Size 000b 128 bytes max payload size	0h	

		001b 256 bytes max payload size 010b 512 bytes max payload size others Reserved for ASM1064/116x		
4	RW	Enable Relaxed Ordering	0	
3	RW	Unsupported Request Reporting Enable	0	
2	RW	Fatal Error Reporting Enable	0	
1	RW	Non-Fatal Error Reporting Enable	0	
0	RW	Correctable Error Reporting Enable	0	

7.5. Device Status Register

PCI CFG offset – 8Ah

Register Name: DEV_STS

Size: 16 bits

Bit	Attrib	Description	Default	Mnemonic
15:6	Rsvd	Reserved	000h	
5	RO	Transactions Pending	0	
4	RO	AUX Power Detected	0	
3	RW1C	Unsupported Request Detected	0	
2	RW1C	Fatal Error Detected	0	
1	RW1C	Non-Fatal Error Detected	0	
0	RW1C	Correctable Error Detected	0	

7.6. Link Capabilities Register

PCI CFG offset – 8Ch

Register Name: LNK_CAP

Size: 32 bits

Bit	Attrib	Description	Default	Mnemonic
31:24	RO	Port Number	0000h	
23	Rsvd	Reserved	0	
22	RO	ASPM Optionality Compliance	1	
21	RO	Link Bandwidth Notification Capability	0	
20	RO	Data Link Layer Link Active Reporting Capable	0	
19	RO	Surprise Down Error Reporting Capable	0	
18	RO	Clock Power Management	1	
17:15	RO	L1 Exit Latency 110b 32 μ s-64 μ s	6h	
14:12	RO	L0s Exit Latency 110b 2 μ s-4 μ s	6h	
11:10	RO	Active State Power Management (ASPM) Support 11b L0s and L1 Supported	3h	
9:4	RO	Maximum Link Width 00 0001b x1 (ASM1062A/1064) 00 0010b x2 (ASM1164/1165/1166)	01h or 02h	
3:0	RO	Max Link Speed 0011b Supported Link Speeds Vector field bit 2	3h	

7.7. Link Control Register

PCI CFG offset – 90h

Register Name: LNK_CTRL

Size: 16 bits

Bit	Attrib	Description	Default	Mnemonic
15:14	Rsvd	DRS Signaling Control Reserved for ASM1064/116x	0	
13:12	Rsvd	Reserved	0	
11	Rsvd	Link Autonomous Bandwidth Interrupt Enable Reserved for ASM1064/116x	0	
10	Rsvd	Link Bandwidth Management Interrupt Enable Reserved for ASM1064/116x	0	
9	RW	Hardware Autonomous Width Disable	0	
8	RW	Enable Clock Power Management	0	
7	RW	Extended Synch	0	
6	RW	Common Clock Configuration	0	
5	Rsvd	Retrain Link Reserved for ASM1064/116x	0	
4	Rsvd	Link Disable Reserved for ASM1064/116x	0	
3	Rsvd	Read Completion Boundary (RCB) Reserved for ASM1064/116x	0	
2	Rsvd	Reserved	0	
1:0	RW	Active State Power Management (ASPM) Control 00b Disabled 01b L0s Entry Enabled 10b L1 Entry Enabled 11b L0s and L1 Entry Enabled	0	

7.8. Link Status Register

PCI CFG offset – 92h

Register Name: LNK_STS

Size: 16 bits

Bit	Attrib	Description	Default	Mnemonic
15	Rsvd	Link Autonomous Bandwidth Status Reserved for ASM1064/116x	0	
14	Rsvd	Link Bandwidth Management Status Reserved for ASM1064/116x	0	
13	Rsvd	Data Link Layer Link Active Reserved for ASM1064/116x	0	
12	RO	Slot Clock Configuration	1	
11	Rsvd	Link Training Reserved for ASM1064/116x	0	
10	RO	Undefined	0	
9:4	RO	Negotiated Link Width 00 0001b x1 00 0010b x2 others Reserved for ASM1064/116x	00h	
3:0	RO	Current Link Speed 0001b Supported Link Speeds Vector field bit 0 0010b Supported Link Speeds Vector field bit 1 0011b Supported Link Speeds Vector field bit 2	0h	

7.9. Link Capabilities 2 Register

PCI CFG offset – 2Ch

Register Name: LNK_CAP2

Size: 32 bits

Bit	Attrib	Description	Default	Mnemonic
31	RO	DRS Supported	0	
30:23	Rsvd	Reserved	00h	
22:16	RO	Lower SKP OS Reception Supported Speeds Vector Bit 0 2.5 GT/s Bit 1 5.0 GT/s Bit 2 8.0 GT/s Bits 6:3 RsvdP	07h	
15:9	RO	Lower SKP OS Generation Supported Speeds Vector Bit 0 2.5 GT/s Bit 1 5.0 GT/s Bit 2 8.0 GT/s Bits 6:3 RsvdP	07h	
8	RO	Crosslink Supported	0	
7:1	RO	Supported Link Speeds Vector Bit 0 2.5 GT/s Bit 1 5.0 GT/s Bit 2 8.0 GT/s Bits 6:3 RsvdP	07h	
0	Rsvd	Reserved	0	

7.10. Advanced Error Reporting (AER) Capability Register

AER Capability Register starts from PCI CFG offset 100h. Next table shows the PCI Express Advanced Error Reporting Capability structure of ASM1064/116x.

PCI CFG
Offset

31	0
100h	Advanced Error Reporting Extended Capability Header
104h	Uncorrectable Error Status Register
108h	Uncorrectable Error Mask Register
10Ch	Uncorrectable Error Severity Register
110h	Correctable Error Status Register
114h	Correctable Error Mask Register
118h	Advanced Error Capabilities and Control Register
11Ch	Header Log Register
120h	
124h	
128h	

7.10.1. Advanced Error Reporting Extended Capability Header

PCI CFG offset – 100h

Register Name: AER_CAPH

Size: 32 bits

Bit	Attrib	Description	Default	Mnemonic
31:20	RO	Next Capability Offset	130h	
19:16	RO	Capability Version	1h	
15:0	RO	PCI Express Extended Capability ID	0001h	

7.10.2. Uncorrectable Error Status Register

PCI CFG offset – 104h

Register Name: AER_UES

Size: 32 bits

Bit	Attrib	Description	Default	Mnemonic
31:27	Rsvd	Reserved	00h	
26	RW1C	Poisoned TLP Egress Blocked Status (Optional)	0	
25	RW1C	TLP Prefix Blocked Error Status (Optional)	0	
24	RW1C	AtomicOp Egress Blocked Status (Optional)	0	
23	RW1C	MC Blocked TLP Status (Optional)	0	
22	RW1C	Uncorrectable Internal Error Status (Optional)	0	
21	RW1C	ACS Violation Status (Optional)	0	
20	RW1C	Unsupported Request Error Status	0	
19	RW1C	ECRC Error Status (Optional)	0	
18	RW1C	Malformed TLP Status	0	
17	RW1C	Receiver Overflow Status (Optional)	0	
16	RW1C	Unexpected Completion Status	0	
15	RW1C	Completer Abort Status (Optional)	0	
14	RW1C	Completion Timeout Status	0	
13	RW1C	Flow Control Protocol Error Status (Optional)	0	
12	RW1C	Poisoned TLP Received Status	0	
11:6	Rsvd	Reserved	00h	
5	RW1C	Surprise Down Error Status (Optional)	0	
4	RW1C	Data Link Protocol Error Status	0	
3:1	Rsvd	Reserved	0h	
0	RO	Undefined	0	

7.10.3. Uncorrectable Error Mask Register

PCI CFG offset – 108h

Register Name: AER_UEM

Size: 32 bits

Bit	Attrib	Description	Default	Mnemonic
31:27	Rsvd	Reserved	00h	
26	RW	Poisoned TLP Egress Blocked Mask (Optional)	1	
25	RW	TLP Prefix Blocked Error Mask (Optional)	0	
24	RW	AtomicOp Egress Blocked Mask (Optional)	0	
22	RW	Uncorrectable Internal Error Mask (Optional)	1	
23	RW	MC Blocked TLP Mask (Optional)	0	
21	RW	ACS Violation Mask (Optional)	0	
20	RW	Unsupported Request Error Mask	0	

19	RW	ECRC Error Mask (Optional)	0	
18	RW	Malformed TLP Mask	0	
17	RW	Receiver Overflow Mask (Optional)	0	
16	RW	Unexpected Completion Mask	0	
15	RW	Completer Abort Mask (Optional)	0	
14	RW	Completion Timeout Mask	0	
13	RW	Flow Control Protocol Error Mask (Optional)	0	
12	RW	Poisoned TLP Received Mask	0	
11:6	Rsvd	Reserved	00h	
5	RW	Surprise Down Error Mask (Optional)	0	
4	RW	Data Link Protocol Error Mask	0	
3:1	Rsvd	Reserved	0h	
0	RO	Undefined	0	

7.10.4. Uncorrectable Error Severity Register

PCI CFG offset – 10Ch

Register Name: AER_UESE

Size: 32 bits

Bit	Attrib	Description	Default	Mnemonic
31:27	Rsvd	Reserved	00h	
26	RW	Poisoned TLP Egress Blocked Severity (Optional)	0	
25	RW	TLP Prefix Blocked Error Severity (Optional)	0	
24	RW	AtomicOp Egress Blocked Severity (Optional)	0	
23	RW	MC Blocked TLP Severity (Optional)	0	
22	RW	Uncorrectable Internal Error Severity (Optional)	1	
21	RW	ACS Violation Severity (Optional)	0	
20	RW	Unsupported Request Error Severity	0	
19	RW	ECRC Error Severity (Optional)	0	
18	RW	Malformed TLP Severity	1	
17	RW	Receiver Overflow Severity (Optional)	1	
16	RW	Unexpected Completion Severity	0	
15	RW	Completer Abort Severity (Optional)	0	
14	RW	Completion Timeout Severity	0	
13	RW	Flow Control Protocol Error Severity (Optional)	1	
12	RW	Poisoned TLP Received Severity	0	
11:6	Rsvd	Reserved	00h	
5	RW	Surprise Down Error Severity (Optional)	1	
4	RW	Data Link Protocol Error Severity	1	
3:1	Rsvd	Reserved	0h	
0	RO	Undefined	0	

7.10.5. Correctable Error Status Register

PCI CFG offset – 110h

Register Name: AER_CES

Size: 32 bits

Bit	Attrib	Description	Default	Mnemonic
-----	--------	-------------	---------	----------

31:16	Rsvd	Reserved	0000h	
15	RW1C	Header Log Overflow Status (Optional)	0	
14	RW1C	Corrected Internal Error Status (Optional)	0	
13	RW1C	Advisory Non-Fatal Error Status	0	
12	RW1C	Replay Timer Timeout Status	0	
11:9	Rsvd	Reserved	0h	
8	RW1C	REPLAY_NUM Rollover Status	0	
7	RW1C	Bad DLLP Status	0	
6	RW1C	Bad TLP Status	0	
5:1	Rsvd	Reserved	00h	
0	RW1C	Receiver Error Status	0	

7.10.6. Correctable Error Mask Register

PCI CFG offset – 114h

Register Name: AER_CEM

Size: 32 bits

Bit	Attrib	Description	Default	Mnemonic
31:16	Rsvd	Reserved	0000h	
15	RW	Header Log Overflow Mask (Optional)	1	
14	RW	Corrected Internal Error Mask (Optional)	1	
13	RW	Advisory Non-Fatal Error Mask	1	
12	RW	Replay Timer Timeout Mask	0	
11:9	Rsvd	Reserved	0h	
8	RW	REPLAY_NUM Rollover Mask	0	
7	RW	Bad DLLP Mask	0	
6	RW	Bad TLP Mask	0	
5:1	Rsvd	Reserved	00h	
0	RW	Receiver Error Mask	0	

7.10.7. Advanced Error Capabilities and Control Register

PCI CFG offset – 118h

Register Name: AER_CCT

Size: 32 bits

Bit	Attrib	Description	Default	Mnemonic
31:13	Rsvd	Reserved	00000h	
12	RO	Completion Timeout Prefix/Header Log Capable	0	
11	RO	TLP Prefix Log Present	0	
10	RW	Multiple Header Recording Enable	0	
9	RO	Multiple Header Recording Capable	0	
8	RW	ECRC Check Enable	0	
7	RO	ECRC Check Capable	0	
6	RW	ECRC Generation Enable	0	
5	RO	ECRC Generation Capable	0	
4:0	RO	First Error Pointer	00h	

7.10.8. Header Log Register

PCI CFG offset 11Ch ~ 12Bh is Header Log Register. The Header Log register contains the header for the TLP corresponding to a detected error. Please refer to PCI Express® Base Specification Revision 3.1 for the detail.

ASMedia Confidential

8. Use I²C to Access Internal Registers

The internal registers can be accessed by I²C bus. Please don't access the reserved area and limited area of different products when use this method. Otherwise, unexpected condition or system crash may be happened.

8.1. Internal Registers Map

There are three Register Spaces – PCIe Configuration Space, Private Control Space, and AHCI Control Space described in section 2. Registers Space. Next table shows the internal registers map.

Address	Register	Note
F000h ~ FFFFh	Reserved	
E000h ~ EFFFh	PCI Configuration Space	
C000h ~ DFFFh	Reserved	
BE00h ~ BFFFh	Reserved	
BD00h ~ BDFFh	GPIO/Timer/LED	
BC00h ~ BCFFh	UART Controller	
BB00h ~ BBFFh	SPI Controller	
BA00h ~ BAFFh	I2C Controller	
B900h ~ B9FFh	Chip configuration	
B800h ~ B8FFh	Reserved	
B700h ~ B7FFh	PCIe Endpoint	
B600h ~ B6FFh	PCIe Transaction layer	
B500h ~ B5FFh	PCIe Link Layer	
B400h ~ B4FFh	PCS MAC	
B100h ~ B3FFh	Reserved	
B080h ~ B0FFh	PCS Lane 1	ASM1166/1165/1164 only
B000h ~ B07Fh	PCS Lane 0	
AC00h ~ AFFFh	Reserved	
AA00h ~ ABFFh	SATA Port 5 Transport/Link/Phy Layer	ASM1166 only
A800h ~ A9FFh	SATA Port 4 Transport/Link/Phy Layer	ASM1166/1165 only
A600h ~ A7FFh	SATA Port 3 Transport/Link/Phy Layer	ASM1166/1165/1164/1064 only
A400h ~ A5FFh	SATA Port 2 Transport/Link/Phy Layer	ASM1166/1165/1164/1064 only
A200h ~ A3FFh	SATA Port 1 Transport/Link/Phy Layer	
A000h ~ A1FFh	SATA Port 0 Transport/Link/Phy Layer	
8000h ~ 9FFFh	Standard AHCI Registers	
0000h ~ 7FFFh	Reserved	

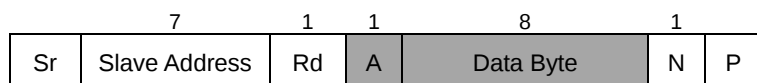
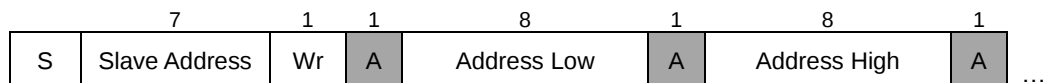
8.2. Read/Write Protocols

A 7-bits slave address is required following an I²C start condition to enable a read or write operation. The default 7-bits slave address is 0001001b and can be modified by requirement. Followings show the Read/Write protocol to read/write a byte of internal registers.

Write Protocol:

	7	1	1	8	1	8	1	8	1	
S	Slave Address	Wr	A	Address Low	A	Address High	A	Data Byte	A	P

Read Protocol:

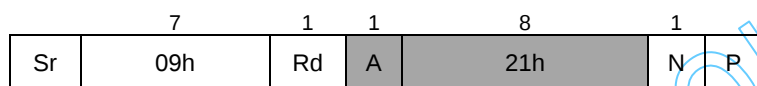
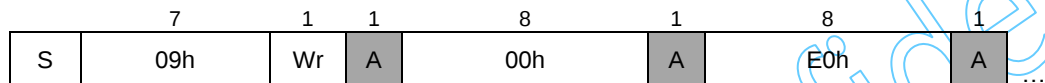


Example: Read PCI CFG offset 0 Vendor ID

1. Read the first byte

Slave Address = 09h, Address Low = 00h, Address High = E0h

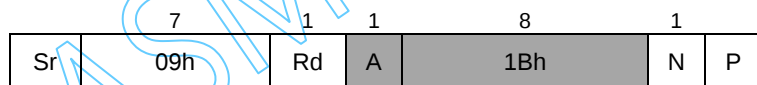
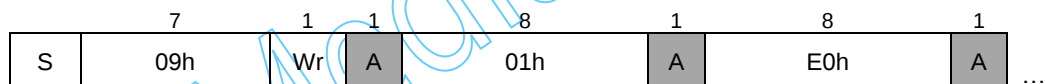
Read Data Byte = 21h



2. Read the second byte

Slave Address = 09h, Address Low = 01h, Address High = E0h

Read Data Byte = 1Bh



9. PCI Memory Space Access Constraint

There are three PCI memory space in ASM1064/116x series controllers:

BAR0 (PCI Rx10 ~ Rx13): Private Control Memory Space

BAR5 (PCI Rx24 ~ Rx27): AHCI Control Memory Space

BAR30 (PCI Rx30 ~ Rx33): Expansion ROM Space

These memory spaces can use Byte/Word/Dword method to access. DO NOT use Qword (64-bits) method to access. Otherwise, wrong result will be returned.

Example: Read expansion ROM to system memory

```
//*****
//Procedure: ReadExpansionRom
//Description: Read expansion rom to system memory
//Input:      pBaseAddr      - PCI expansion rom base address (PCI BAR30 Rx30~33)
//            pMemAddr       - System memory address to store expansion rom
//            BlockNumber    - Expansion rom length in 512 bytes block
//Output:      None
//Note:
//*****
VOID ReadExpansionRom(PVOID pBaseAddr, PVOID pMemAddr, UINT BlockNumber)
{
    PDWORD    pSrc;
    PDWORD    pDst;
    ULONG     Length;    //length in bytes
    UINT      i;

    pSrc = (PDWORD)pBaseAddr;
    pDst = (PDWORD)pMemAddr;
    Length = BlockNumber * 512;

    for( i = 0; i < (Length / sizeof(DWORD)); i++ )
    {
        pDst[i] = pSrc[i];
    }

    return;
}
```